Palm OS® Programmer's API Reference

Palm OS® 68K SDK

Written by Greg Wilson, Jean Ostrem, Christopher Bey, and Mark Dugger
Technical assistance from Jesse Donaldson, Noah Gibbs, Lee Taylor, Danny Epstein, Peter Epstein, Ludovic Ferrandis, Gilles Fabre, David Fedor, Roger Flores, Steve Lemke, Bob Ebert, Ken Krugler, Paul Plaquette, Andy Stewart, Bruce Thompson, Tim Wiegman, Gavin Peacock, Ryan Robertson, and Waddah Kudaimi

Copyright © 1996-2004, PalmSource, Inc. and its affiliates. All rights reserved. This technical documentation contains confidential and proprietary information of PalmSource, Inc. ("PalmSource"), and is provided to the licensee ("you") under the terms of a Nondisclosure Agreement, Product Development Kit license, Software Development Kit license or similar agreement between you and PalmSource. You must use commercially reasonable efforts to maintain the confidentiality of this technical documentation. You may print and copy this technical documentation solely for the permitted uses specified in your agreement with PalmSource. In addition, you may make up to two (2) copies of this technical documentation for archival and backup purposes. All copies of this technical documentation remain the property of PalmSource, and you agree to return or destroy them at PalmSource's written request. Except for the foregoing or as authorized in your agreement with PalmSource, you may not copy or distribute any part of this technical documentation in any form or by any means without express written consent from PalmSource, Inc., and you may not modify this technical documentation or make any derivative work of it (such as a translation, localization, transformation or adaptation) without express written consent from PalmSource.

PalmSource, Inc. reserves the right to revise this technical documentation from time to time, and is not obligated to notify you of any revisions.

THIS TECHNICAL DOCUMENTATION IS PROVIDED ON AN "AS IS" BASIS. NEITHER PALMSOURCE NOR ITS SUPPLIERS MAKES, AND EACH OF THEM EXPRESSLY EXCLUDES AND DISCLAIMS TO THE FULL EXTENT ALLOWED BY APPLICABLE LAW, ANY REPRESENTATIONS OR WARRANTIES REGARDING THIS TECHNICAL DOCUMENTATION, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING WITHOUT LIMITATION ANY WARRANTIES IMPLIED BY ANY COURSE OF DEALING OR COURSE OF PERFORMANCE AND ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, ACCURACY, AND SATISFACTORY QUALITY. PALMSOURCE AND ITS SUPPLIERS MAKE NO REPRESENTATIONS OR WARRANTIES THAT THIS TECHNICAL DOCUMENTATION IS FREE OF ERRORS OR IS SUITABLE FOR YOUR USE. TO THE FULL EXTENT ALLOWED BY APPLICABLE LAW, PALMSOURCE, INC. ALSO EXCLUDES FOR ITSELF AND ITS SUPPLIERS ANY LIABILITY, WHETHER BASED IN CONTRACT OR TORT (INCLUDING NEGLIGENCE), FOR DIRECT, INCIDENTAL, CONSEQUENTIAL, INDIRECT, SPECIAL, EXEMPLARY OR PUNITIVE DAMAGES OF ANY KIND ARISING OUT OF OR IN ANY WAY RELATED TO THIS TECHNICAL DOCUMENTATION, INCLUDING WITHOUT LIMITATION DAMAGES FOR LOST REVENUE OR PROFITS, LOST BUSINESS, LOST GOODWILL, LOST INFORMATION OR DATA, BUSINESS INTERRUPTION, SERVICES STOPPAGE, IMPAIRMENT OF OTHER GOODS, COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, OR OTHER FINANCIAL LOSS, EVEN IF PALMSOURCE, INC. OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR IF SUCH DAMAGES COULD HAVE BEEN REASONABLY FORESEEN.

PalmSource, the PalmSource logo, BeOS, Graffiti, HandFAX, HandMAIL, HandPHONE, HandSTAMP, HandWEB, HotSync, the HotSync logo, iMessenger, MultiMail, MyPalm, Palm, the Palm logo, the Palm trade dress, Palm Computing, Palm OS, Palm Powered, PalmConnect, PalmGear, PalmGlove, PalmModem, Palm Pack, PalmPak, PalmPix, PalmPower, PalmPrint, Palm.Net, Palm Reader, Palm Talk, Simply Palm and ThinAir are trademarks of PalmSource, Inc. or its affiliates. All other product and brand names may be trademarks or registered trademarks of their respective owners.

IF THIS TECHNICAL DOCUMENTATION IS PROVIDED ON A COMPACT DISC, THE SOFTWARE AND OTHER DOCUMENTATION ON THE COMPACT DISC ARE SUBJECT TO THE LICENSE AGREEMENTS ACCOMPANYING THE SOFTWARE AND OTHER DOCUMENTATION.

Palm OS Programmer's API Reference
Document Number 3003-011
November 9, 2004
For the latest version of this document, visit
http://www.palmos.com/dev/support/docs/.

PalmSource, Inc.
1240 Crossman Avenue
Sunnyvale, CA 94089
USA
www.palmsource.com

Table of Contents

About This Document **lxxxv**

Palm OS SDK Documentation lxxxv
What This Volume Contains lxxxv
Additional Resources lxxxvi
Conventions Used in This Guide lxxxvii

Part I: User Interface

1 Application Launch Codes **3**

Launch Codes 6
sysAppLaunchCmdAddRecord 6
sysAppLaunchCmdAlarmTriggered 9
sysAppLaunchCmdAttention 10
sysAppLaunchCmdCardLaunch 14
sysAppLaunchCmdCountryChange 16
sysAppLaunchCmdDisplayAlarm 16
sysAppLaunchCmdExgAskUser 17
sysAppLaunchCmdExgGetData 19
sysAppLaunchCmdExgPreview 19
sysAppLaunchCmdExgReceiveData 21
sysAppLaunchCmdFind 22
sysAppLaunchCmdGoTo 25
sysAppLaunchCmdGoToURL 27
sysAppLaunchCmdHandleSyncCallApp 28
sysAppLaunchCmdInitDatabase 30
sysAppLaunchCmdLookup 31
sysAppLaunchCmdNormalLaunch 31
sysAppLaunchCmdNotify 32
sysAppLaunchCmdOpenDB 32
sysAppLaunchCmdPanelCalledFromApp 33
sysAppLaunchCmdReturnFromPanel 33
sysAppLaunchCmdSaveData 34

sysAppLaunchCmdSyncNotify 34
sysAppLaunchCmdSystemLock 35
sysAppLaunchCmdSystemReset 35
sysAppLaunchCmdTimeChange 36
sysAppLaunchCmdURLParams 36
Launch Flags 37

2 Palm OS Events 41

Event Data Structures 42
eventsEnum 42
EventType 43
EventPtr 45
Event Reference 45
appStopEvent 45
attnIndicatorEnterEvent 46
attnIndicatorSelectEvent 46
ctlEnterEvent 46
ctlExitEvent 47
ctlRepeatEvent 47
ctlSelectEvent 48
daySelectEvent 49
fldChangedEvent 50
fldEnterEvent 50
fldHeightChangedEvent 51
frmCloseEvent 52
frmGadgetEnterEvent 52
frmGadgetMiscEvent 53
frmGotoEvent 53
frmLoadEvent 54
frmObjectFocusLostEvent 55
frmObjectFocusTakeEvent 56
frmOpenEvent 56
frmSaveEvent 57
frmTitleEnterEvent 57
frmTitleSelectEvent 57

frmUpdateEvent 58
inetSockReadyEvent 59
inetSockStatusChangeEvent 59
keyDownEvent 60
keyHoldEvent 62
. 63
. 63
keyUpEvent 63
lstEnterEvent 64
lstExitEvent 64
lstSelectEvent 65
menuCloseEvent 65
menuCmdBarOpenEvent 66
menuEvent 66
menuOpenEvent 67
nilEvent 68
penDownEvent 68
penMoveEvent 69
penUpEvent 69
popSelectEvent 70
sclEnterEvent 71
sclExitEvent 71
sclRepeatEvent 72
tblEnterEvent 73
tblExitEvent 73
tblSelectEvent 74
winDisplayChangedEvent 75
winEnterEvent 75
winExitEvent 76

3 Notifications **79**

Notification Data Structures 82
SysNotifyParamType 82
Notification Reference 84
cncNotifyProfileEvent 84

kTelTelephonyNotification 85
Socket Notification 86
sysExternalConnectorAttachEvent 87
sysExternalConnectorDetachEvent 87
sysNotifyAntennaRaisedEvent 88
sysNotifyAppLaunchingEvent 88
sysNotifyAppQuittingEvent 89
sysNotifyCardInsertedEvent 89
sysNotifyCardRemovedEvent 90
sysNotifyDBCreatedEvent 91
sysNotifyDBChangedEvent 92
sysNotifyDBDeletedEvent 95
sysNotifyDBDirtyEvent 96
sysNotifyDeleteProtectedEvent 97
sysNotifyDeviceUnlocked 98
sysNotifyDisplayChangeEvent 99
sysNotifyDisplayResizedEvent 100
sysNotifyEarlyWakeupEvent 100
sysNotifyEventDequeuedEvent 101
sysNotifyForgotPasswordEvent 101
sysNotifyGotUsersAttention 102
sysNotifyGsiDrawIndicator 102
sysNotifyHelperEvent 103
sysNotifyIdleTimeEvent 104
sysNotifyInputAreaDrawingEvent 104
sysNotifyInputAreaPendownEvent 105
sysNotifyInsPtEnableEvent 106
sysNotifyKeyboardDialogEvent 106
sysNotifyLateWakeupEvent 107
sysNotifyLocaleChangedEvent 108
sysNotifyMenuCmdBarOpenEvent 108
sysNotifyNetLibIFMediaEvent 109
sysNotifyProcessPenStrokeEvent 110
sysNotifyResetFinishedEvent 111
sysNotifyRetryEnqueueKey 111

sysNotifySelectDay 111
sysNotifySleepNotifyEvent 113
sysNotifySleepRequestEvent 113
sysNotifySyncFinishEvent 115
sysNotifySyncStartEvent 115
sysNotifyTimeChangeEvent 116
sysNotifyVirtualCharHandlingEvent 116
sysNotifyVolumeMountedEvent 117
sysNotifyVolumeUnmountedEvent 117

4 Attention Manager 119

Attention Manager Data Structures 119
AttnCommandType 119
AttnCommandArgsType 121
. 121
AttnFlagsType 124
AttnLaunchCodeArgsType 126
AttnLevelType 127
Attention Manager Constants 128
Error Code Constants 128
Attention Manager Drawing Constants 129
Attention Manager Feature Constants 129
Attention Manager Functions 131
AttnDoSpecialEffects 131
AttnForgetIt 132
AttnGetAttention 133
AttnGetCounts. 135
AttnIndicatorEnable 136
AttnIndicatorEnabled 137
AttnIterate 137
AttnListOpen 138
AttnUpdate 139
Application-Defined Functions 141
AttnCallbackProc 141

5 Categories **145**
Category Data Structures . . . 145
AppInfoPtr . . . 145
AppInfoType . . . 145
Category Constants . . . 146
Category Functions . . . 147
CategoryCreateList . . . 147
CategoryCreateListV10 . . . 149
CategoryEdit . . . 150
CategoryEditV20 . . . 151
CategoryEditV10 . . . 152
CategoryFind . . . 152
CategoryFreeList . . . 153
CategoryFreeListV10 . . . 154
CategoryGetName . . . 154
CategoryGetNext. . . . 155
CategoryInitialize . . . 156
CategorySelect . . . 157
CategorySelectV10 . . . 159
CategorySetName . . . 160
CategorySetTriggerLabel . . . 160
CategoryTruncateName . . . 161

6 Clipboard **163**
Clipboard Data Structures . . . 163
ClipboardFormatType . . . 163
Clipboard Functions . . . 164
ClipboardAddItem . . . 164
ClipboardAppendItem . . . 165
ClipboardGetItem . . . 166

7 Controls **167**
Control Data Structures . . . 167
ButtonFrameType . . . 167
ControlAttrType . . . 168
ControlPtr . . . 169

ControlStyleType 169
ControlType 171
GraphicControlType 172
SliderControlType 174
Control Resources 176
Control Functions. 176
CtlDrawControl 176
CtlEnabled 177
CtlEraseControl 177
CtlGetLabel 178
CtlGetSliderValues 178
CtlGetValue 179
CtlHandleEvent 180
CtlHideControl 181
CtlHitControl 181
CtlNewControl 182
CtlNewGraphicControl 184
CtlNewSliderControl 185
CtlSetEnabled 187
CtlSetGraphics 187
CtlSetLabel 188
CtlSetSliderValues 189
CtlSetUsable 190
CtlSetValue 190
CtlShowControl 191
CtlValidatePointer 192

8 Date and Time Selector 195

Date and Time Selections Data Structures 195
SelectDayType 195
DaySelectorType 195
HMSTime 196
Date and Time Selection Functions 196
DayDrawDays 196
DayDrawDaySelector 197

DayHandleEvent 197
SelectDay 198
SelectDayV10 199
SelectOneTime 199
SelectTime 200
SelectTimeV33 201
SelectTimeZone 202

9 Fields **205**

Field Data Structures 205
FieldAttrType 205
FieldPtr 208
FieldType 208
LineInfoPtr 212
LineInfoType 212
Field Resources 213
Field Functions 213
FldCalcFieldHeight 213
FldCompactText 214
FldCopy 214
FldCut 215
FldDelete 215
FldDirty 216
FldDrawField 217
FldEraseField 217
FldFreeMemory 218
FldGetAttributes 218
FldGetBounds 219
FldGetFont 219
FldGetInsPtPosition 220
FldGetMaxChars 220
FldGetNumberOfBlankLines 221
FldGetScrollPosition 221
FldGetScrollValues 222
FldGetSelection 223

FldGetTextAllocatedSize 223
FldGetTextHandle 224
FldGetTextHeight 225
FldGetTextLength 225
FldGetTextPtr 226
FldGetVisibleLines 226
FldGrabFocus 227
FldHandleEvent 227
FldInsert 229
FldMakeFullyVisible 230
FldNewField 231
FldPaste. 233
FldRecalculateField. 233
FldReleaseFocus 234
FldScrollable. 235
FldScrollField 235
FldSendChangeNotification 236
FldSendHeightChangeNotification 237
FldSetAttributes 237
FldSetBounds 238
FldSetDirty 239
FldSetFont. 239
FldSetInsertionPoint 240
FldSetInsPtPosition. 241
FldSetMaxChars 241
FldSetMaxVisibleLines 242
FldSetScrollPosition 242
FldSetSelection. 243
FldSetText 244
FldSetTextAllocatedSize. 246
FldSetTextHandle 246
FldSetTextPtr 248
FldSetUsable 249
FldUndo 250
FldWordWrap 250

10 Find **253**
Find Functions 253
FindDrawHeader 253
FindGetLineBounds 254
FindSaveMatch 254
FindStrInStr 256

11 Forms **259**
Form Structures and Types 259
AlertTemplateType 259
FormActiveStateType 260
FormAttrType 261
FormBitmapType 262
FormFrameType 263
FormGadgetAttrType 263
FormGadgetType 265
FormGadgetTypeInCallback 267
FormLabelType 268
FormLineType 269
FormObjAttrType 269
FormObjectType 270
FormObjListType 272
FormPopupType 272
FormPtr 273
FormRectangleType 273
FormTitleType 274
FormType 274
FrmGraffitiStateType 276
FrmNavHeaderType 277
FrmNavOrderEntryType 278
Form Constants 279
AlertType 279
FormObjectKind 280
FrmNavFocusRingStyleEnum 281
FrmNavHeaderFlagsType 282

FrmNavObjectFlagsType 284
FrmNavStateFlagsType 285
Form Navigation Focus Error Codes 286
Form Dynamic Input Area Policies 287
Miscellaneous Form Constants 287
One-Handed Navigation Trap Selectors 288
Form Resources 289
Form Functions. 290
ECFrmValidatePtr 290
FrmAlert 290
FrmCloseAllForms 291
FrmCopyLabel. 291
FrmCopyTitle 292
FrmCountObjectsInNavOrder 293
FrmCustomAlert 293
FrmCustomResponseAlert. 294
FrmDeleteForm 295
FrmDispatchEvent 296
FrmDoDialog 296
FrmDrawForm. 297
FrmEraseForm 298
FrmGetActiveField 298
FrmGetActiveForm. 299
FrmGetActiveFormID. 299
FrmGetControlGroupSelection 299
FrmGetControlValue 300
FrmGetDIAPolicyAttr 301
FrmGetFirstForm. 301
FrmGetFocus 301
FrmGetFormBounds 302
FrmGetFormId. 302
FrmGetFormPtr 302
FrmGetGadgetData. 303
FrmGetLabel 303
FrmGetNavEntry. 304

FrmGetNavOrder 305
FrmGetNavState 306
FrmGetNumberOfObjects 306
FrmGetObjectBounds 307
FrmGetObjectId 307
FrmGetObjectIndex. 308
FrmGetObjectIndexFromPtr 309
FrmGetObjectPosition 310
FrmGetObjectPtr 310
FrmGetObjectType 311
FrmGetTitle 311
FrmGetWindowHandle 311
FrmGotoForm 312
FrmHandleEvent. 312
FrmHelp 316
FrmHideObject 316
FrmInitForm. 317
FrmNavDrawFocusRing 317
FrmNavGetFocusRingInfo. 319
FrmNavObjectTakeFocus 320
FrmNavRemoveFocusRing 321
FrmNewBitmap 322
FrmNewForm 323
FrmNewGadget 324
FrmNewGsi 325
FrmNewLabel 326
FrmPointInTitle 327
FrmPopupForm 327
FrmRemoveObject 328
FrmRestoreActiveState 329
FrmReturnToForm 329
FrmSaveActiveState 330
FrmSaveAllForms 330
FrmSetActiveForm 331
FrmSetCategoryLabel. 331

FrmSetControlGroupSelection 332
FrmSetControlValue 332
FrmSetDIAPolicyAttr 333
FrmSetEventHandler 334
FrmSetFocus. 334
FrmSetGadgetData 335
FrmSetGadgetHandler 336
FrmSetMenu. 336
FrmSetNavEntry 337
FrmSetNavOrder. 338
FrmSetNavState 339
FrmSetObjectBounds 340
FrmSetObjectPosition 340
FrmSetTitle 341
FrmShowObject 341
FrmUpdateForm 342
FrmUpdateScrollers 343
FrmValidatePtr. 344
FrmVisible 344
Application-Defined Functions 345
FormCheckResponseFuncType. 345
FormEventHandlerType. 346
FormGadgetHandlerType 346

12 Graffiti Shift **349**
GraffitiShift Functions. 349
GsiEnable 349
GsiEnabled 349
GsiInitialize 350
GsiSetLocation. 350
GsiSetShiftState 350

13 Insertion Point **353**
Insertion Point Functions 353
InsPtEnable 353
InsPtEnabled 354

InsPtGetHeight 354
InsPtGetLocation 354
InsPtSetHeight 355
InsPtSetLocation 355

14 Lists **357**

List Data Structures 357
ListAttrType 357
ListType 358
List Resources 361
List Functions 361
LstDrawList 361
LstEraseList 362
LstGetNumberOfItems 362
LstGetSelection 363
LstGetSelectionText 363
LstGetTopItem 364
LstGetVisibleItems 364
LstHandleEvent 364
LstMakeItemVisible 365
LstNewList 366
LstPopupList 367
LstScrollList 367
LstSetDrawFunction 368
LstSetHeight 368
LstSetListChoices 369
LstSetPosition 370
LstSetSelection 370
LstSetTopItem 371
Application-Defined Function 371
ListDrawDataFuncType 372

15 Menus **377**

Menu Data Structures 377
MenuBarAttrType 377
MenuCmdBarButtonType 378

MenuCmdBarResultType 380
MenuCmdBarType 380
MenuBarPtr 382
MenuBarType 382
MenuItemType 384
MenuPullDownPtr 385
MenuPullDownType 385
Menu Constants 387
Menu Resources 387
Menu Functions 387
MenuAddItem 387
MenuCmdBarAddButton 389
MenuCmdBarDisplay 392
MenuCmdBarGetButtonData 393
MenuDispose 394
MenuDrawMenu 394
MenuEraseStatus 395
MenuGetActiveMenu 397
MenuHandleEvent 397
MenuHideItem 399
MenuInit 400
MenuSetActiveMenu 400
MenuSetActiveMenuRscID 401
MenuShowItem 401

16 Private Records 403

Private Record Data Structures 403
privateRecordViewEnum 403
Private Record Functions 404
SecSelectViewStatus 404
SecVerifyPW 404

17 Progress Manager 407

Progress Manager Functions 407
PrgHandleEvent 407
PrgStartDialog 408

PrgStartDialogV31 409
PrgStopDialog 410
PrgUpdateDialog. 411
PrgUserCancel 412
Application-Defined Functions 412
PrgCallbackFunc 412

18 Scroll Bars **417**
Scroll Bar Data Structures 417
ScrollBarAttrType 417
ScrollBarPtr 418
ScrollBarType 419
Scroll Bar Resources. 421
Scroll Bar Functions 421
SclDrawScrollBar. 421
SclGetScrollBar 421
SclHandleEvent 422
SclSetScrollBar 423

19 System Dialogs **427**
System Dialog Functions 427
SysAppLauncherDialog 427
SysFatalAlert 428
SysGraffitiReferenceDialog 428

20 Tables **429**
Table Data Structures 429
TableAttrType 429
TableColumnAttrType 430
TableItemPtr 432
TableItemType 432
TablePtr 438
TableRowAttrType 438
TableType 440
Table Constants 442
Table Resource 443

Table Functions 443
TblDrawTable 443
TblEditing 444
TblEraseTable 445
TblFindRowData 445
TblFindRowID 446
TblGetBounds 446
TblGetColumnSpacing 447
TblGetColumnWidth 447
TblGetCurrentField 448
TblGetItemBounds 448
TblGetItemFont 449
TblGetItemInt 449
TblGetItemPtr 450
TblGetLastUsableRow 451
TblGetNumberOfColumns 451
TblGetNumberOfRows 452
TblGetRowData 452
TblGetRowHeight 453
TblGetRowID 453
TblGetSelection 454
TblGetTopRow 454
TblGrabFocus 455
TblHandleEvent 456
TblHasScrollBar 457
TblInsertRow 458
TblMarkRowInvalid 458
TblMarkTableInvalid 459
TblRedrawTable 459
TblReleaseFocus 460
TblRemoveRow 461
TblRowInvalid 462
TblRowMasked 462
TblRowSelectable 463
TblRowUsable 463

TblSelectItem 464
TblSetBounds 465
TblSetColumnEditIndicator 465
TblSetColumnMasked 466
TblSetColumnSpacing 467
TblSetColumnUsable 467
TblSetColumnWidth 468
TblSetCustomDrawProcedure 468
TblSetItemFont 469
TblSetItemInt 470
TblSetItemPtr 471
TblSetItemStyle 472
TblSetLoadDataProcedure 473
TblSetRowData 474
TblSetRowHeight 474
TblSetRowID 475
TblSetRowMasked 475
TblSetRowSelectable 476
TblSetRowStaticHeight 477
TblSetRowUsable 478
TblSetSaveDataProcedure 478
TblSetSelection. 479
TblUnhighlightSelection 480
Application-Defined Functions 480
TableDrawItemFuncType 480
TableLoadDataFuncType 481
TableSaveDataFuncType 483

21 UI Color List **485**
UI Color Data Types 485
UIColorTableEntries 485
UI Color Functions 489
UIColorGetTableEntryIndex 489
UIColorGetTableEntryRGB 490
UIColorSetTableEntry. 491

22 UI Controls **493**
UI Control Functions 493
UIBrightnessAdjust. 493
UIContrastAdjust 493
UIPickColor 494

23 Miscellaneous User Interface Functions **497**
Miscellaneous User Interface Data Structures 497
AddressLookupFields 497
AddrLookupParamsType 498
Miscellaneous User Interface Functions 501
PhoneNumberLookup 501
PhoneNumberLookupCustom 502
ResLoadConstant 503
ResLoadForm 504
ResLoadMenu 505

Part II: System Management

24 Alarm Manager **509**
Alarm Manager Functions 509
AlmGetAlarm 509
AlmGetProcAlarm 510
AlmSetAlarm 510
AlmSetProcAlarm 512
Application-Defined Functions 513
AlmAlarmProcPtr 513

25 ARM Debug Nub **515**
ARM Debug Nub Structures and Types 516
AdnDbgPostLoadParamsType 516
AdnDbgPostUnloadParamsType 516
ARM Debug Nub Constants 516
Debugging Feature Flags 516
Semihosting SWI Opcodes 517

Miscellaneous ARM Debug Nub Constants 518
ARM Debug Nub Functions and Macros 519
AdnDebugBreak 519
AdnDebugEnableGet 519
AdnDebugEnableGetSupported 520
AdnDebugEnableSet 520
AdnDebugLicenseeSpecific 521
AdnDebugMessage 521
AdnDebugMessageIf 522
AdnDebugNativeRegister 522
AdnDebugNativeRegisterAddr 523
AdnDebugNativeUnregister 524
AdnDebugNativeUnregisterAddr 524
AdnDebugUpdateLoadedModules 525
AdnGetNativeCodeBaseAddr 525
SetR10ToDataPointer 526

26 Bitmaps **527**

Bitmap Data Structures 527
BitmapCompressionType 527
BitmapDirectInfoType 528
BitmapFlagsType 529
BitmapPtr 532
BitmapType 532
BitmapTypeV0 534
BitmapTypeV1 535
BitmapTypeV2 537
BitmapTypeV3 539
ColorTableType 542
DensityType 543
PixelFormatType 544
RGBColorType 545
Bitmap Constants 546
Bitmap Resources 547
Bitmap Functions 547

BmpBitsSize 547
BmpColortableSize 548
BmpCompress 548
BmpCreate 549
BmpCreateBitmapV3 551
BmpDelete 552
BmpGetBits 553
BmpGetBitDepth 553
BmpGetColortable 554
BmpGetCompressionType 554
BmpGetDensity 555
BmpGetDimensions 555
BmpGetNextBitmap 556
BmpGetNextBitmapAnyDensity 557
BmpGetSizes 558
BmpGetTransparentValue 559
BmpGetVersion 559
BmpSetDensity 560
BmpSetTransparentValue 560
BmpSize 561
ColorTableEntries 562

27 Byte Order Utilities 567

ByteOrderUtils Functions and Macros 567
EndianSwap16 567
EndianSwap32 567
Read68KUnaligned32 568
Write68KUnaligned32 568

28 Certificate Manager 571

Certificate Manager Structures and Types 572
CertMgrCertChainType 572
CertMgrCertElementEnum 572
CertMgrCertFieldEnum 573
CertMgrCertInfoType 573
CertMgrCertSearchEnum 574

CertMgrElementListType 574
CertMgrElementType 575
CertMgrVerifyResultType 576
Certificate Manager Constants 576
X509Cert Element Fields 576
RSA Element Fields 578
RDN Element Fields 579
X509Extensions Element Fields 579
Data Types 580
Certificate Formats 581
Certificate Manager Error Codes 581
Certificate Verification Failure Codes 583
Miscellaneous Certificate Manager Constants 584
Certificate Manager Element Field Macros 584
apCertMgrElementFieldRDNOIDN 584
apCertMgrElementFieldRDNValueN 585
apCertMgrElementFieldX509ExBytesN 585
apCertMgrElementFieldX509ExCriticalN 585
apCertMgrElementFieldX509ExOIDN 586
Certificate Manager Functions and Macros 586
CertMgrAddCert. 586
CertMgrExportCert. 588
CertMgrFindCert. 590
CertMgrGetField 591
CertMgrImportCert 593
CertMgrReleaseCertInfo 595
CertMgrRemoveCert 595
CertMgrVerifyCert 596
CertMgrVerifyFailure 597

29 Character Attributes 599

Character Attribute Functions 599
ChrHorizEllipsis 599
ChrIsHardKey 599
ChrNumericSpace 600

GetCharAttr 600
GetCharCaselessValue 601
GetCharSortValue 602

30 Data and Resource Manager 603

Data Manager Data Structures 603
DmOpenRef 603
DmResID 603
DmResType 604
SortRecordInfoType 604
Data Manager Constants. 604
Category Constants 604
Record Attribute Constants 605
Database Attribute Constants 606
Error Codes 608
Open Mode Constants 611
Miscellaneous Constants 612
Data Manager Functions. 612
DmArchiveRecord 612
DmAttachRecord. 613
DmAttachResource 615
DmCloseDatabase 616
DmCreateDatabase 617
DmCreateDatabaseFromImage. 619
DmDatabaseInfo 619
DmDatabaseProtect 621
DmDatabaseSize 622
DmDeleteCategory 623
DmDeleteDatabase 625
DmDeleteRecord 626
DmDetachRecord 627
DmDetachResource. 628
DmFindDatabase. 629
DmFindRecordByID 629
DmFindResource. 630

DmFindResourceType 631
DmFindSortPosition 632
DmFindSortPositionV10 633
DmGetAppInfoID 634
DmGetDatabase 635
DmGetDatabaseLockState 635
DmGetLastErr 636
DmGetNextDatabaseByTypeCreator 637
DmGetRecord 640
DmGetResource 641
DmGetResourceIndex. 642
DmGet1Resource. 643
DmInsertionSort 643
DmMoveCategory 645
DmMoveRecord 646
DmNewHandle 647
DmNewRecord 648
DmNewResource 649
DmNextOpenDatabase 650
DmNextOpenResDatabase 650
DmNumDatabases 651
DmNumRecords 651
DmNumRecordsInCategory 652
DmNumResources 653
DmOpenDatabase 653
DmOpenDatabaseByTypeCreator 656
DmOpenDatabaseInfo 657
DmOpenDBNoOverlay 658
DmPositionInCategory 659
DmQueryNextInCategory 660
DmQueryRecord 661
DmQuickSort 662
DmRecordInfo 663
DmReleaseRecord 664
DmReleaseResource 665

DmRemoveRecord 665
DmRemoveResource 666
DmRemoveSecretRecords 667
DmResizeRecord 667
DmResizeResource 668
DmResourceInfo 669
DmSearchRecord 670
DmSearchResource 670
DmSeekRecordInCategory 671
DmSet 672
DmSetDatabaseInfo 673
DmSetRecordInfo 675
DmSetResourceInfo 676
DmStrCopy 677
DmSync. 678
DmSyncDatabase 678
DmWrite 679
DmWriteCheck 680
Application-Defined Functions 680
DmComparF 680

31 Debug Manager 683

Debug Manager Structures and Types 683
DbgCtlEnumInfoType 683
DbgCtlHandlerInfoType 684
Debug Manager Constants 685
DbgControl() Constants 685
System-Defined Debug Control Operations 686
Debug Manager Functions and Macros 687
DbgBreak 687
DbgCommSettings 687
DbgGetMessage 688
DbgInit 688
DbgMessage. 689
DbgSrcBreak. 689

DbgSrcMessage 689
Application-Defined Functions 690
DbgControlFuncType 690
DbgCtlEnumCallbackFunc 691

32 Error Manager 693
ERROR_CHECK_LEVEL Define 693
Error Manager Data Structures 694
ErrExceptionType 694
Error Manager Functions and Macros 694
ErrAlert 694
ErrCatch 695
ErrDisplay 696
ErrDisplayFileLineMsg 697
ErrEndCatch 697
ErrExceptionList 698
ErrFatalDisplayIf 698
ErrNonFatalDisplayIf 699
ErrThrow 699
ErrTry 700

33 Expansion Manager 701
Expansion Manager Data Structures. 701
ExpCardInfoType 701
Expansion Manager Constants 702
Error Codes 702
Defined Media Types 703
Expansion Manager Functions 705
ExpCardGetSerialPort 705
ExpCardInfo. 706
ExpCardPresent 706
ExpSlotDriverInstall 707
ExpSlotDriverRemove 708
ExpSlotEnumerate 709
ExpSlotLibFind 710

34 Feature Manager **713**
Feature Constants. 713
Networking Features 714
Expansion and VFS Manager Features 716
Hardware Capability Features 717
Hardware Features 720
Emulator and Debugging Features 725
Localization Features 726
Miscellaneous Features 729
Feature Manager Functions and Macros 730
FtrGet 730
FtrGetByIndex 731
FtrPtrFree 732
FtrPtrNew. 732
FtrPtrResize 734
FtrSet 735
FtrUnregister 735

35 File Streaming **737**
File Streaming Constants 737
Primary Open Mode Constants 737
Secondary Open Mode Constants 738
File Streaming Functions. 738
FileClearerr 738
FileClose 739
FileControl 739
FileDelete 745
FileDmRead 745
FileEOF 746
FileError 747
FileFlush 747
FileGetLastError 748
FileOpen 748
FileRead 750
FileRewind 751

FileSeek 752
FileTell 753
FileTruncate 753
FileWrite 754
File Streaming Error Codes. 755

36 Float Manager **757**
Float Manager Data Structures 757
FlpCompDouble 757
FlpDoubleBits 758
Float Manager Functions 759
FlpAToF. 759
FlpBase10Info 760
FlpBufferAToF 761
FlpBufferCorrectedAdd 762
FlpBufferCorrectedSub 763
FlpCorrectedAdd 764
FlpCorrectedSub 765
FlpFToA 766
FlpGetExponent 766
FlpGetSign 767
FlpIsZero 767
FlpNegate 767
FlpSetNegative 768
FlpSetPositive 768
FlpVersion. 769

37 Fonts **771**
Font Data Structures 771
FontCharInfoPtr 771
FontCharInfoType 771
FontDensityType 772
FontID 773
FontPtr 774
FontType 774
FontTypeV2 776

Font Constants 779
Font Resources 779
Font Resource 779
Extended Font Resource 782
Font Functions 785
FntAverageCharWidth 785
FntBaseLine 785
FntCharHeight 785
FntCharsInWidth 786
FntCharsWidth 787
FntCharWidth 787
FntDefineFont 788
FntDescenderHeight 789
FntGetFont 789
FntGetFontPtr 790
FntGetScrollValues 790
FntIsAppDefined 791
FntLineHeight 791
FntLineWidth 791
FntSetFont 792
FntWCharWidth 793
FntWidthToOffset 793
FntWordWrap 794
FntWordWrapReverseNLines 795
FontSelect 796

38 Graffiti Manager 797

Graffiti Manager Functions 797
GrfAddMacro 797
GrfAddPoint 798
GrfCleanState 798
GrfDeleteMacro 798
GrfFilterPoints 799
GrfFindBranch 799
GrfFlushPoints 800

GrfGetAndExpandMacro 800
GrfGetGlyphMapping 801
GrfGetMacro 801
GrfGetMacroName 802
GrfGetNumPoints 802
GrfGetPoint 803
GrfGetState 803
GrfInitState 804
GrfMatch 804
GrfMatchGlyph 805
GrfProcessStroke 806
GrfSetState 806

39 Helper API **809**
Helper Data Structures 809
HelperNotifyEnumerateListType 809
HelperNotifyEventType 810
HelperNotifyExecuteType 812
HelperNotifyValidateType 813
HelperServiceEMailDetailsType 814
HelperServiceSMSDetailsType 815
Helper Constants 815
Helper Service Class IDs 815

40 Key Manager **819**
Key Manager Functions 819
KeyCurrentState 819
KeyRates 820
KeySetMask 820

41 Locale Manager **823**
Locale Manager Data Types 823
CountryType 823
LanguageType 823
LmLocaleType 824
NumberFormatType 824

Locale Manager Constants 825
Character Encoding Constants 825
Country Constants 827
Language Constants 827
Locale Manager Size Constants 828
Locale Manager Functions 828
LmGetLocaleSetting 828
LmGetNumLocales 832
LmLocaleToIndex 833

42 Memory Manager 835

Memory Manager Functions 835
MemCardInfo 835
MemCmp 836
MemDebugMode 837
MemHandleCardNo 837
MemHandleDataStorage 837
MemHandleFree 838
MemHandleHeapID 838
MemHandleLock 838
MemHandleNew 839
MemHandleResize 839
MemHandleSetOwner 840
MemHandleSize 841
MemHandleToLocalID 841
MemHandleUnlock 842
MemHeapCheck 842
MemHeapCompact 842
MemHeapDynamic 843
MemHeapFlags 843
MemHeapFreeBytes 844
MemHeapID 845
MemHeapScramble 845
MemHeapSize 846
MemLocalIDKind 846

MemLocalIDToGlobal 846
MemLocalIDToLockedPtr 847
MemLocalIDToPtr 847
MemMove 848
MemNumCards 848
MemNumHeaps 849
MemNumRAMHeaps 849
MemPtrCardNo 849
MemPtrDataStorage 850
MemPtrFree 850
MemPtrHeapID 851
MemPtrNew. 851
MemPtrRecoverHandle 852
MemPtrResize 852
MemPtrSetOwner 853
MemPtrSize 853
MemPtrToLocalID 854
MemPtrUnlock 854
MemSet. 854
MemSetDebugMode 855
MemStoreInfo 856

43 Notification Manager 859

Notification Constants. 859
Miscellaneous Constants 859
Notification Functions. 860
SysNotifyBroadcast. 860
SysNotifyBroadcastDeferred 861
SysNotifyBroadcastFromInterrupt 862
SysNotifyRegister 864
SysNotifyUnregister 866
Application-Defined Functions 867
SysNotifyProcPtr. 867

44 Overlay Manager 869

Overlay Manager Data Structures. 869

OmLocaleType 869
Overlay Manager Constants 870
Overlay Manager Functions 871
OmGetCurrentLocale 871
OmGetIndexedLocale 872
OmGetNextSystemLocale 873
OmGetRoutineAddress 874
OmGetSystemLocale 874
OmLocaleToOverlayDBName 875
OmOverlayDBNameToLocale 876
OmSetSystemLocale 877

45 Password **879**
Password Functions 879
PwdExists 879
PwdRemove 879
PwdSet 880
PwdVerify 880

46 Pen Input Manager Reference **881**
Pen Input Manager Constants 881
Input Area States 881
Pen Input Manager Functions 884
PINGetInputAreaState 884
PINGetInputTriggerState 884
PINSetInputAreaState 885
PINSetInputTriggerState 886
Control Bar Functions 886
StatBarGetIcon 887
StatBarSetIcon 887
StatGetAttribute 888
StatHide 889
StatShow 889
Display Orientation Functions 890
SysGetOrientation 890
SysGetOrientationTriggerState 891

SysSetOrientation 891
SysSetOrientationTriggerState 892

47 Pen Manager **895**
Pen Manager Functions 895
PenCalibrate. 895
PenResetCalibration 896

48 Preferences **897**
Preferences Data Types 897
MeasurementSystemType 897
SecurityAutoLockType 897
SoundLevelTypeV20 898
SystemPreferencesChoice 899
Preferences Constants 910
Preferences Functions 912
PrefGetAppPreferences 912
PrefGetAppPreferencesV10 914
PrefGetPreference 915
PrefGetPreferences 915
PrefOpenPreferenceDB 916
PrefOpenPreferenceDBV10 916
PrefSetAppPreferences 917
PrefSetAppPreferencesV10 918
PrefSetPreference. 919
PrefSetPreferences 919

49 Rectangles **921**
Rectangle Data Structures 921
PointType 921
RectangleType 921
Rectangle Functions. 922
RctCopyRectangle 922
RctGetIntersection 922
RctInsetRectangle 923
RctOffsetRectangle 924

RctPtInRectangle 925
RctSetRectangle 925

50 Sound Manager **927**

Simple Sound Data Structures 927
SndCallbackInfoType 927
SndCommandType 928
SndCmdIDType 929
SndMidiListItemType 929
SndMidiRecHdrType 930
SndMidiRecType 930
SndSmfCallbacksType 931
SndSmfChanRangeType 931
SndSmfCmdEnum 932
SndSmfOptionsType 932
SndSysBeepType 933
Simple Sound Constants 933
SndCmdIDTag 933
Simple Sound Amplitudes 934
sndMidiNameLength 934
sndMidiRecSignature 935
SndPlaySmf Commands 935
sndSmfPlayAllMilliSec 935
System Beeps 935
Simple Sound Functions 937
SndCreateMidiList 937
SndDoCmd 937
SndGetDefaultVolume 940
SndInterruptSmfIrregardless 941
SndPlaySmf 941
SndPlaySmfIrregardless 944
SndPlaySmfResource 946
SndPlaySmfResourceIrregardless 947
SndPlaySystemSound 948
SndSetDefaultVolume 948

Simple Sound Application-Defined Functions 949
SndBlockingFuncType 949
SndComplFuncType 950
Sampled Sound Data Structures 950
SndFormatType 950
SndPtr 951
SndSampleType 951
SndStreamMode 951
SndStreamRef 951
SndStreamWidth 952
Sampled Sound Constants 952
Sound Stream Feature Constants 952
Sound Data Encoding Formats 952
SndSampleTag 953
SndStreamModeTag 955
SndStreamWidthTag 956
Sound Resource Playback Flags 956
Stereo Pan Constants 957
Volume Constants 957
Sampled Sound Functions 958
SndPlayResource 958
SndStreamCreate 960
SndStreamCreateExtended 964
SndStreamDelete 966
SndStreamGetPan 967
SndStreamGetVolume 967
SndStreamPause 968
SndStreamSetPan 968
SndStreamSetVolume 969
SndStreamStart 970
SndStreamStop 971
Sampled Sound Application-Defined Functions 971
SndStreamBufferCallback 971
SndStreamVariableBufferCallback 973

51 Standard IO 977

Standard IO Functions and Macros . . . 977
fgetc . . . 977
fgets . . . 978
fprintf . . . 978
fputc . . . 979
fputs . . . 979
getchar . . . 980
gets . . . 980
printf . . . 980
putc . . . 981
putchar . . . 981
puts . . . 982
SioAddCommand . . . 982
Siofgetc . . . 983
Siofgets . . . 983
Siofprintf . . . 984
Siofputc . . . 984
Siofputs . . . 985
Siogets . . . 985
Sioprintf . . . 986
Sioputs . . . 986
Siosystem . . . 987
Siovfprintf. . . . 987
sprintf . . . 988
system . . . 988
vfprintf . . . 989
vsprintf . . . 990
Standard IO Provider Functions . . . 990
SioClearScreen . . . 990
SioExecCommand . . . 991
SioFree . . . 991
SioHandleEvent . . . 992
SioInit . . . 992
Application-Defined Function . . . 992

SioMain 993

52 String Manager **995**

String Manager Functions 995
StrAToI 995
StrCaselessCompare 995
StrCat. 996
StrChr 996
StrCompare 997
StrCompareAscii 998
StrCopy 999
StrDelocalizeNumber 999
StrIToA 1000
StrIToH 1001
StrLen 1001
StrLocalizeNumber 1002
StrNCaselessCompare 1003
StrNCat 1003
StrNCompare 1005
StrNCompareAscii 1006
StrNCopy 1006
StrPrintF 1008
StrStr 1008
StrToLower 1009
StrVPrintF. 1010

53 System Event Manager **1015**

System Event Manager Data Structures 1015
System Event Manager Functions 1015
EvtAddEventToQueue 1015
EvtAddUniqueEventToQueue 1016
EvtCopyEvent 1017
EvtDequeuePenPoint 1017
EvtDequeuePenStrokeInfo. 1018
EvtEnableGraffiti. 1018
EvtEnqueueKey 1019

EvtEventAvail 1019
EvtFlushKeyQueue 1020
EvtFlushNextPenStroke 1020
EvtFlushPenQueue 1020
EvtGetEvent 1021
EvtGetPen 1021
EvtGetPenBtnList 1022
EvtGetPenNative 1023
EvtGetSilkscreenAreaList 1024
EvtKeydownIsVirtual 1025
EvtKeyQueueEmpty 1025
EvtKeyQueueSize 1026
EvtPenQueueSize 1026
EvtProcessSoftKeyStroke 1026
EvtResetAutoOffTimer 1027
EvtSetAutoOffTimer 1028
EvtSetNullEventTick 1029
EvtSysEventAvail 1029
EvtWakeup 1030
EvtWakeupWithoutNilEvent 1030

54 System Manager **1031**

System Manager Data Structures 1031
SysDBListItemType 1031
System Functions 1032
SysAppLaunch 1032
SysBatteryInfo 1033
SysBatteryInfoV20 1034
SysBinarySearch 1036
SysBroadcastActionCode 1037
SysCopyStringResource 1038
SysCreateDataBaseList 1038
SysCreatePanelList 1040
SysCurAppDatabase 1041
SysErrString 1042

SysFormPointerArrayToStrings 1042
sysFtrNumProcessorIs68K 1043
sysFtrNumProcessorIsARM 1044
SysGetOSVersionString 1044
SysGetROMToken 1045
SysGetStackInfo 1046
SysGetTrapAddress. 1046
SysGremlins 1047
SysHandleEvent 1048
SysInsertionSort 1048
SysKeyboardDialog 1050
SysKeyboardDialogV10 1050
SysLibFind 1051
SysLibInstall. 1051
SysLibLoad 1052
SysLibRemove 1053
SysQSort 1053
SysRandom 1055
SysReset 1055
SysSetAutoOffTime. 1056
SysSetTrapAddress 1056
SysStringByIndex 1057
SysTaskDelay 1058
SysTicksPerSecond 1058
SysUIAppSwitch 1058
Application-Defined Functions 1060
PilotMain 1060

55 Text Manager **1061**

Text Manager Data Structures 1061
CharEncodingType 1061
TranslitOpType 1062
TxtConvertStateType 1062
Text Manager Constants 1063
Character Encoding Feature Attributes 1064

Character Attributes 1065
Character Sizes 1066
Text Manager Functions 1067
TxtByteAttr 1067
TxtCaselessCompare 1068
TxtCharAttr 1070
TxtCharBounds 1071
TxtCharEncoding 1072
TxtCharIsAlNum 1073
TxtCharIsAlpha 1073
TxtCharIsCntrl 1074
TxtCharIsDelim 1074
TxtCharIsDigit 1075
TxtCharIsGraph 1075
TxtCharIsHardKey 1076
TxtCharIsHex 1076
TxtCharIsLower 1077
TxtCharIsPrint 1077
TxtCharIsPunct 1078
TxtCharIsRockerKey 1078
TxtCharIsSpace 1079
TxtCharIsUpper 1079
TxtCharIsValid 1080
TxtCharSize 1080
TxtCharWidth 1081
TxtCharXAttr 1081
TxtCompare 1082
TxtConvertEncoding 1084
TxtEncodingName 1092
TxtFindString 1093
TxtGetChar 1094
TxtGetNextChar 1095
TxtGetPreviousChar 1096
TxtGetTruncationOffset 1097
TxtGetWordWrapOffset 1098

TxtMaxEncoding 1099
TxtNameToEncoding 1099
TxtNextCharSize 1100
TxtParamString 1101
TxtPreviousCharSize 1102
TxtReplaceStr 1103
TxtSetNextChar 1104
TxtStrEncoding 1105
TxtTransliterate 1106
TxtWordBounds 1108

56 Text Services Manager 1111

Text Services Manager Data Structures. 1111
TsmFepModeType 1111
Text Services Manager Functions 1112
TsmGetFepMode 1112
TsmSetFepMode 1113

57 Time Manager 1115

Time Manager Data Structures 1115
DateFormatType 1115
DateTimeType 1118
DateType 1119
DaylightSavingsTypes 1119
DayOfMonthType 1120
TimeFormatType 1123
TimeType 1124
Time Manager Constants 1125
Time Manager Functions 1125
DateAdjust 1125
DateDaysToDate 1126
DateSecondsToDate 1126
DateTemplateToAscii 1127
DateToAscii 1131
DateToDays 1132
DateToDOWDMFormat 1132

DayOfMonth 1133
DayOfWeek 1134
DaysInMonth 1135
TimAdjust 1135
TimDateTimeToSeconds 1136
TimGetSeconds 1136
TimGetTicks 1137
TimSecondsToDateTime 1137
TimSetSeconds 1138
TimeToAscii 1138
TimeZoneToAscii 1139
TimTimeZoneToUTC 1140
TimUTCToTimeZone 1141

58 Virtual File System Manager 1143

VFS Manager Data Structures 1143
FileInfoType 1143
FileRef 1144
VFSAnyMountParamType 1144
VFSSlotMountParamType 1145
VFSPOSEMountParamType 1145
VolumeInfoType 1146
VFS Manager Constants 1147
Defined File Systems 1147
Open Mode Constants 1148
File and Directory Attributes 1149
Volume Attributes 1150
Volume Mount Classes 1150
Error Codes 1151
VFS Manager Functions 1153
VFSCustomControl 1153
VFSDirCreate 1154
VFSDirEntryEnumerate 1156
VFSExportDatabaseToFile 1158
VFSExportDatabaseToFileCustom 1159

VFSFileClose 1161
VFSFileCreate 1162
VFSFileDBGetRecord 1163
VFSFileDBGetResource 1165
VFSFileDBInfo 1166
VFSFileDelete 1169
VFSFileEOF 1170
VFSFileGetAttributes 1171
VFSFileGetDate 1171
VFSFileOpen 1173
VFSFileRead 1174
VFSFileReadData. 1175
VFSFileRename 1177
VFSFileResize 1178
VFSFileSeek 1179
VFSFileSetAttributes 1181
VFSFileSetDate 1182
VFSFileSize 1183
VFSFileTell 1184
VFSFileWrite 1185
VFSGetDefaultDirectory 1186
VFSImportDatabaseFromFile 1187
VFSImportDatabaseFromFileCustom 1189
VFSInstallFSLib 1191
VFSRegisterDefaultDirectory 1192
VFSRemoveFSLib 1194
VFSUnregisterDefaultDirectory 1194
VFSVolumeEnumerate 1195
VFSVolumeFormat 1197
VFSVolumeGetLabel 1199
VFSVolumeInfo 1200
VFSVolumeMount 1201
VFSVolumeSetLabel 1204
VFSVolumeSize 1205
VFSVolumeUnmount 1206

Application-Defined Functions 1207
VFSExportProcPtr 1207
VFSImportProcPtr 1208

59 Windows **1211**

Window Data Structures 1211
CustomPatternType 1211
DrawStateFlagsType 1211
DrawStateType 1212
FrameBitsType 1215
FrameType 1216
IndexedColorType 1217
PatternType 1218
UnderlineModeType 1219
WindowFlagsType 1219
WindowFormatType 1221
WindowType 1222
WinDrawOperation 1224
WinHandle 1226
WinLineType 1226
WinPtr 1227
Window Constants 1227
Window Coordinate System Constants 1227
Window Functions 1228
WinClipRectangle 1228
WinCopyRectangle 1229
WinCreateBitmapWindow 1230
WinCreateOffscreenWindow 1231
WinCreateWindow 1233
WinDeleteWindow 1234
WinDisplayToWindowPt 1235
WinDrawBitmap 1236
WinDrawChar 1237
WinDrawChars 1238
WinDrawGrayLine 1239

WinDrawGrayRectangleFrame 1240
WinDrawInvertedChars 1240
WinDrawLine 1241
WinDrawPixel 1242
WinDrawRectangle 1242
WinDrawRectangleFrame 1243
WinDrawTruncChars 1244
WinEraseChars 1245
WinEraseLine 1246
WinErasePixel 1246
WinEraseRectangle 1247
WinEraseRectangleFrame 1247
WinEraseWindow 1248
WinFillLine 1248
WinFillRectangle 1249
WinGetActiveWindow 1249
WinGetBitmap 1250
WinGetBounds 1250
WinGetClip 1251
WinGetCoordinateSystem 1251
WinGetDisplayExtent 1252
WinGetDisplayWindow 1252
WinGetDrawWindow 1253
WinGetDrawWindowBounds 1253
WinGetFirstWindow 1253
WinGetFramesRectangle 1254
WinGetPattern 1254
WinGetPatternType 1255
WinGetPixel 1255
WinGetPixelRGB 1256
WinGetScalingMode 1257
WinGetSupportedDensity 1257
WinGetWindowExtent 1258
WinGetWindowFrameRect 1259
WinIndexToRGB 1259

WinInvertChars 1260
WinInvertLine 1260
WinInvertPixel 1261
WinInvertRectangle 1261
WinInvertRectangleFrame 1262
WinModal 1263
WinPaintBitmap 1263
WinPaintChar 1264
WinPaintChars 1265
WinPaintLine 1266
WinPaintLines 1266
WinPaintPixel 1267
WinPaintPixels 1268
WinPaintRectangle 1268
WinPaintRectangleFrame 1269
WinPaintRoundedRectangleFrame 1270
WinPaintTiledBitmap 1271
WinPalette 1271
WinPopDrawState 1273
WinPushDrawState 1274
WinResetClip 1274
WinRestoreBits 1275
WinRGBToIndex 1275
WinSaveBit 1276
WinScaleCoord 1277
WinScalePoint 1278
WinScaleRectangle 1279
WinScreenGetAttribute 1280
WinScreenLock 1282
WinScreenMode 1283
WinScreenUnlock 1287
WinScrollRectangle 1287
WinSetActiveWindow 1288
WinSetBackColor 1288
WinSetBackColorRGB 1289

WinSetBounds 1290
WinSetClip 1290
WinSetConstraintsSize 1291
WinSetCoordinateSystem 1292
WinSetDrawMode 1293
WinSetDrawWindow 1293
WinSetForeColor 1294
WinSetForeColorRGB 1294
WinSetPattern 1295
WinSetPatternType 1296
WinSetScalingMode 1296
WinSetTextColor 1299
WinSetTextColorRGB 1300
WinSetUnderlineMode 1301
WinUnscaleCoord 1301
WinUnscalePoint 1302
WinUnscaleRectangle 1304
WinValidateHandle 1304
WinWindowToDisplayPt 1305

60 Miscellaneous System Functions 1307

Crc16CalcBlock 1307
DlkControl 1308
DlkGetSyncInfo 1311
IntlGetRoutineAddress 1314
IntlSetRoutineAddress 1315
LocGetNumberSeparators 1316
PceNativeCall 1316

Part III: Communications

61 Connection Manager 1323

Connection Manager Data Types 1323
CncProfileID 1323

Connection Manager Constants 1324
General Profile Parameters 1324
Phone Profile Parameters 1329
GPRS Profile Parameters 1331
Profile Parameter Sizes 1332
Profile Parameter Types 1334
Device Kinds 1335
Connection Manager Functions 1336
CncAddProfile 1336
CncDefineParamID 1338
CncDeleteProfile 1339
CncGetParamType 1340
CncGetProfileInfo 1340
CncGetProfileList 1342
CncGetSystemFlagBitnum 1343
CncGetTrueParamID 1343
CncIsFixedLengthParamType 1344
CncIsSystemFlags 1344
CncIsSystemRange 1345
CncIsThirdPartiesRange 1345
CncIsVariableLengthParamType 1346
CncProfileCloseDB 1346
CncProfileCount 1347
CncProfileCreate 1347
CncProfileDelete 1348
CncProfileGetCurrent 1348
CncProfileGetIDFromIndex 1349
CncProfileGetIDFromName 1350
CncProfileGetIndex 1350
CncProfileOpenDB 1351
CncProfileSetCurrent 1352
CncProfileSettingGet 1352
CncProfileSettingSet 1353

62 Exchange Manager 1355

Exchange Manager Data Structures 1355
ExgAskResultType 1355
ExgGoToType 1356
ExgLocalSocketInfoType 1356
ExgPreviewInfoType 1358
ExgSocketType 1360
Exchange Manager Constants 1363
Registry ID Constants 1363
Predefined URL Schemes 1364
Predefined URL Prefixes 1365
Exchange Manager Functions 1366
ExgAccept. 1366
ExgConnect 1367
ExgControl 1369
ExgDBRead 1371
ExgDBWrite 1373
ExgDisconnect 1375
ExgDoDialog 1377
ExgGet 1380
ExgGetDefaultApplication 1381
ExgGetRegisteredApplications 1383
ExgGetRegisteredTypes 1384
ExgGetTargetApplication 1385
ExgNotifyGoto. 1388
ExgNotifyPreview 1389
ExgNotifyReceive 1390
ExgPut 1393
ExgReceive 1394
ExgRegisterDatatype 1396
ExgRegisterData 1400
ExgRequest 1401
ExgSend 1402
ExgSetDefaultApplication 1403
Application-Defined Functions 1405

ExgDBDeleteProcPtr 1405
ExgDBReadProcPtr 1406
ExgDBWriteProcPtr. 1407

63 Exchange Library 1409

Exchange Library Functions 1409
ExgLibAccept 1409
ExgLibClose 1410
ExgLibConnect 1410
ExgLibControl 1412
ExgLibDisconnect 1413
ExgLibGet. 1414
ExgLibHandleEvent 1415
ExgLibOpen 1416
ExgLibPut. 1416
ExgLibReceive 1418
ExgLibRequest. 1419
ExgLibSend 1420
ExgLibSleep 1421
ExgLibWake 1422

64 IR Library 1423

IR Library Data Structures 1423
IrConnect 1423
IrPacket 1424
IrIASObject 1425
IrIasQuery 1426
IrCallbackParms 1427
IrStatsType 1428
IR Library Constants 1429
IR Control Constants 1429
IR Stack Callback Events. 1431
LEVENT_DATA_IND 1432
LEVENT_DISCOVERY_CNF 1432
LEVENT_LAP_CON_CNF 1432
LEVENT_LAP_CON_IND 1432

LEVENT_LAP_DISCON_IND 1432
LEVENT_LM_CON_CNF 1432
LEVENT_LM_CON_IND 1433
LEVENT_LM_DISCON_IND 1433
LEVENT_PACKET_HANDLED 1433
LEVENT_STATUS_IND 1433
LEVENT_TEST_CNF 1433
LEVENT_TEST_IND 1434
IR Library Functions 1434
IrAdvanceCredit 1434
IrBind 1435
IrClose 1436
IrConnectIrLap 1436
IrConnectReq 1437
IrConnectRsp 1438
IrDataReq 1439
IrDisconnectIrLap 1440
IrDiscoverReq 1441
IrIsIrLapConnected 1441
IrIsMediaBusy 1442
IrIsNoProgress 1442
IrIsRemoteBusy 1442
IrLocalBusy 1443
IrMaxRxSize 1443
IrMaxTxSize 1444
IrOpen 1445
IrSetConTypeLMP 1445
IrSetConTypeTTP 1446
IrSetDeviceInfo 1446
IrTestReq 1447
IrUnbind 1448
IAS Functions and Macros 1448
IrIAS_Add 1449
IrIAS_GetInteger 1450
IrIAS_GetIntLsap 1450

IrIAS_GetObjectID 1451
IrIAS_GetOctetString 1451
IrIAS_GetOctetStringLen 1451
IrIAS_GetType 1452
IrIAS_GetUserString 1452
IrIAS_GetUserStringCharSet 1452
IrIAS_GetUserStringLen 1453
IrIAS_Next 1453
IrIAS_Query 1454
IrIAS_SetDeviceName 1455
IrIAS_StartResult 1456
Application-Defined Functions 1456
IrIasQueryCallBack 1456

65 Modem Manager 1457

Modem Manager Functions 1457
MdmDial 1457
MdmHangUp 1458

66 Net Library 1459

Net Library Data Structures 1459
NetConfigNameType 1459
NetHostInfoBufType 1460
NetHostInfoType 1460
NetServInfoBufType 1461
NetServInfoType 1462
NetSocketAddrEnum 1462
NetSocketAddrINType 1463
NetSocketAddrRawType 1463
NetSocketAddrType 1464
NetSocketNoticeType 1464
NetSocketRef 1465
NetSocketTypeEnum 1465
NoticeTypeEnum 1466
Net Library Constants 1467
Configuration Aliases 1467

I/O Flags 1468
Socket Notice Trigger Conditions 1468
Tracing Bits 1469
Net Library Functions 1470
NetHToNL 1470
NetHToNS 1471
NetLibAddrAToIN 1471
NetLibAddrINToA 1472
NetLibClose 1472
NetLibConfigAliasGet 1474
NetLibConfigAliasSet 1475
NetLibConfigDelete 1476
NetLibConfigIndexFromName 1477
NetLibConfigList 1478
NetLibConfigMakeActive 1479
NetLibConfigRename 1480
NetLibConfigSaveAs 1481
NetLibConnectionRefresh 1483
NetLibDmReceive 1484
NetLibFinishCloseWait 1485
NetLibGetHostByAddr 1486
NetLibGetHostByName 1488
NetLibGetMailExchangeByName 1490
NetLibGetServByName 1492
NetLibIFAttach 1493
NetLibIFDetach 1494
NetLibIFDown 1495
NetLibIFGet 1496
NetLibIFSettingGet 1497
NetLibIFSettingSet 1504
NetLibIFUp 1505
NetLibMaster 1506
NetLibOpen 1510
NetLibOpenConfig 1511
NetLibOpenCount 1513

NetLibReceive 1514
NetLibReceivePB 1516
NetLibSelect 1518
NetLibSend 1520
NetLibSendPB 1523
NetLibSettingGet 1525
NetLibSettingSet 1529
NetLibSocketAccept 1530
NetLibSocketAddr 1532
NetLibSocketBind 1533
NetLibSocketClose 1535
NetLibSocketConnect 1536
NetLibSocketListen 1537
NetLibSocketOpen 1539
NetLibSocketOptionGet 1541
NetLibSocketOptionSet 1543
NetLibSocketShutdown 1547
NetLibTracePrintF 1548
NetLibTracePutS 1549
NetNToHL 1550
NetNToHS 1550

67 Network Utilities **1551**

Network Utility Functions 1551
NetUReadN 1551
NetUTCPOpen 1552
NetUWriteN 1553

68 Script Plugin **1555**

Script Plugin Data Types 1555
PluginCallbackProcType 1555
PluginCmdPtr 1556
PluginCmdType 1556
PluginExecCmdType 1556
PluginInfoPtr 1557
PluginInfoType 1557

ScriptPluginLaunchCodesEnum 1558
Script Plugin Constants 1559
Command Constants 1559
Size Constants 1560
Script Plugin Functions 1561
ScriptPluginSelectorProc 1561

69 Virtual Drivers 1567

Driver Data Structures. 1567
DrvrInfoType 1567
DrvrRcvQType 1568
DrvrStatusEnum 1570
SrmRcvQType 1570
VdrvAPIType 1571
VdrvConfigType 1572
VdrvCtlOpCodeEnum 1574
Driver Constants 1579
Driver Version Constants 1579
Port Feature Constants 1579
Virtual Driver-Defined Functions 1580
DrvEntryPointProcPtr 1581
VdrvCloseProcPtr 1582
VdrvControlProcPtr 1583
VdrvControlCustomProcPtr 1585
VdrvOpenProcPtr 1586
VdrvOpenProcV4Ptr 1587
VdrvStatusProcPtr 1588
VdrvWriteProcPtr 1588
Serial Manager Queue Functions 1589
GetSizeProcPtr 1589
GetSpaceProcPtr 1589
SignalCheckPtr 1590
WriteBlockProcPtr 1590
WriteByteProcPtr 1591

70 Serial Manager 1593

Serial Manager Data Structures 1593
DeviceInfoType 1593
SrmCtlEnum 1594
SrmOpenConfigType 1597
Serial Manager Constants 1598
Port Constants 1598
Serial Capabilities Constants 1601
Serial Settings Constants 1601
Status Constants 1604
Line Error Constants 1604
Serial Manager Functions 1605
SrmClearErr 1605
SrmClose 1605
SrmControl 1606
SrmCustomControl 1609
SrmExtOpen 1610
SrmExtOpenBackground 1612
SrmGetDeviceCount 1613
SrmGetDeviceInfo 1614
SrmGetStatus 1615
SrmOpen 1616
SrmOpenBackground 1617
SrmPrimeWakeupHandler 1618
SrmReceive 1619
SrmReceiveCheck 1620
SrmReceiveFlush 1621
SrmReceiveWait 1622
SrmReceiveWindowClose 1623
SrmReceiveWindowOpen 1624
SrmSend 1625
SrmSendCheck 1626
SrmSendFlush 1627
SrmSendWait 1628
SrmSetReceiveBuffer 1629

SrmSetWakeupHandler 1630
Serial Manager Application-Defined Functions 1631
WakeupHandlerProcPtr 1631

71 Old Serial Manager 1633
Serial Manager Data Structures 1633
SerCtlEnum 1633
SerSettingsType 1635
Serial Manager Functions 1635
SerClearErr 1635
SerClose 1636
SerControl. 1637
SerGetSettings 1638
SerGetStatus 1638
SerOpen 1639
SerReceive 1641
SerReceive10 1642
SerReceiveCheck 1643
SerReceiveFlush 1643
SerReceiveWait 1644
SerSend 1645
SerSend10 1646
SerSendFlush 1647
SerSendWait 1647
SerSetReceiveBuffer 1648
SerSetSettings 1648

72 Serial Link Manager 1651
Serial Link Manager Functions 1651
SlkClose 1651
SlkCloseSocket. 1652
SlkFlushSocket. 1652
SlkOpen 1653
SlkOpenSocket. 1653
SlkReceivePacket. 1654
SlkSendPacket 1656

SlkSetSocketListener 1656
SlkSocketPortID 1658
SlkSocketSetTimeout 1658

73 Telephony Basic Services 1659

Telephony Service Types 1659
Telephony Data Structures 1661
TelAppID 1661
TelEventType 1662
TelGetCallStateType 1664
TelInfGetInformationType 1665
TelNotificationType 1666
TelOemCallType 1667
TelSendCommandStringType 1667
TelSndPlayKeyToneType 1668
Telephony Constants 1669
Battery Status Constants 1669
Keycode Constants 1669
Telephone Call State Constants 1670
Telephone Call Type Constants 1670
Telephone Call Service Type Constants 1671
Error Code Constants 1671
Information Type Constants 1675
Message Identifier Constants 1675
TelServices 1676
Notification Priorities 1677
Call Classes 1677
Telephony Manager Database 1678
Version Constants 1678
Events and Notifications 1679
Miscellaneous Constants 1679
Notification Identifiers 1680
Telephony Functions 1681
TelCancel 1681
TelClose 1683

TelClosePhoneConnection 1683
TelGetCallState 1684
TelGetEvent 1686
TelGetTelephonyEvent 1687
TelInfGetInformation 1688
TelIsCfgServiceAvailable 1689
TelIsDtcServiceAvailable 1691
TelIsEmcServiceAvailable 1692
TelIsInfServiceAvailable 1693
TelIsNwkServiceAvailable 1694
TelIsOemServiceAvailable 1695
TelIsPhbServiceAvailable 1696
TelIsPhoneConnected 1697
TelIsPowServiceAvailable 1698
TelIsSmsServiceAvailable 1699
TelIsSndServiceAvailable 1701
TelIsSpcServiceAvailable 1702
TelIsStyServiceAvailable 1703
TelMatchPhoneDriver. 1704
TelOemCall 1705
TelOpen. 1706
TelOpenPhoneConnection 1707
TelOpenProfile 1708
TelPowGetBatteryStatus. 1709
TelPowGetPowerLevel 1711
TelPowSetPhonePower 1712
TelSendCommandString 1713
TelSndMute 1714
TelSndPlayKeyTone 1715
TelSndStopKeyTone 1717
TelUnblockNotifications. 1718
Feature Support Functions 1718
TelIsFunctionSupported 1718
TelIs<FunctionName>Supported 1720
TelIsServiceAvailable 1721

TelIs<ServiceSet>Available 1723

74 Telephony Security and Configuration 1727

Telephony Security and Configuration Data Structures . . . 1727
TelCfgGetPhoneNumberType 1728
TelCfgGetSmsCenterType 1728
TelStyChangeAuthenticationType 1729
Telephony Security and Configuration Constants 1730
Authentication State Constants 1730
Telephony Security and Configuration Functions 1730
TelCfgGetPhoneNumber 1731
TelCfgGetSmsCenter 1732
TelCfgSetSmsCenter 1734
TelStyChangeAuthenticationCode 1735
TelStyEnterAuthenticationCode 1737
TelStyGetAuthenticationState 1738

75 Telephony Network 1741

Telephony Network Data Structures. 1741
TelNwkGetLocationType 1741
TelNwkGetNetworkNameType 1742
TelNwkGetNetworksType 1743
Telephony Network Constants 1744
Network Type Constants 1744
Network Search Mode Constants 1744
Telephony Network Functions 1745
TelNwkGetLocation 1745
TelNwkGetNetworkName. 1746
TelNwkGetNetworks 1749
TelNwkGetNetworkType 1750
TelNwkGetSearchMode 1751
TelNwkGetSelectedNetwork. 1753
TelNwkGetSignalLevel 1754
TelNwkSelectNetwork 1756
TelNwkSetSearchMode 1757

76 Telephony Calls **1759**
Telephony Calls Data Structures 1759
TelDtcCallNumberType 1760
TelDtcReceiveDataType 1760
TelDtcSendDataType 1761
TelEmcGetNumberType 1761
TelEmcSetNumberType 1762
TelSpcGetCallerNumberType 1763
TelSpcPlayDTMFType 1763
Telephony Calls Functions 1764
TelDtcCallNumber 1764
TelDtcCloseLine 1765
TelDtcReceiveData 1767
TelDtcSendData 1768
TelEmcCall 1769
TelEmcCloseLine 1771
TelEmcGetNumber 1772
TelEmcGetNumberCount 1774
TelEmcSelectNumber 1775
TelEmcSetNumber 1777
TelSpcAcceptCall 1778
TelSpcCallNumber 1779
TelSpcCloseLine 1782
TelSpcConference 1783
TelSpcGetCallerNumber 1785
TelSpcHoldLine 1787
TelSpcPlayDTMF 1788
TelSpcRejectCall 1789
TelSpcRetrieveHeldLine 1790
TelSpcSelectLine 1791
TelSpcSendBurstDTMF 1793
TelSpcStartContinuousDTMF 1794
TelSpcStopContinuousDTMF 1796

77 Telephony SMS 1799

Telephony SMS Data Structures. 1799
TelSmsDateTimeType 1799
TelSmsDeleteMessageType 1800
TelSmsDeliveryAdvancedCDMAType 1801
TelSmsDeliveryAdvancedGSMType 1802
TelSmsDeliveryAdvancedTDMAType 1803
TelSmsDeliveryMessageType 1805
TelSmsExtensionType 1808
TelSmsGetAvailableStorageType 1809
TelSmsGetMessageCountType 1810
TelSmsManualAckType 1810
TelSmsMultiPartExtensionType 1812
TelSmsNbsExtensionType 1813
TelSmsReadMessagesType 1813
TelSmsReadReportsType 1814
TelSmsReadSubmittedMessagesType 1815
TelSmsReportType 1815
TelSmsSendMessageType 1817
TelSmsSubmitAdvancedCDMAType 1818
TelSmsSubmitAdvancedGSMType 1819
TelSmsSubmitAdvancedTDMAType 1820
TelSmsSubmitMessageType 1820
TelSmsSubmittedMessageType 1822
TelSmsUserExtensionType 1823
Telephony SMS Constants 1823
SMS Extension Type Constants 1823
SMS Message Type Constants 1824
SMS Message Transport Protocol Constants 1824
SMS Storage ID Constants 1825
SMS Data Coding Scheme Constants 1825
SMS Message Urgency Constants 1826
SMS Message Privacy Constants 1826
SMS Message Types 1826
SMS Delivery Status Reports 1827

SMS Report Types 1828
Telephony SMS Functions 1828
TelSmsDeleteMessage. 1829
TelSmsGetAvailableStorage 1830
TelSmsGetDataMaxSize 1832
TelSmsGetMessageCount 1833
TelSmsGetSelectedStorage 1835
TelSmsGetUniquePartId. 1836
TelSmsReadMessage 1838
TelSmsReadMessages 1840
TelSmsReadReport 1842
TelSmsReadReports 1844
TelSmsReadSubmittedMessage. 1846
TelSmsReadSubmittedMessages 1848
TelSmsSelectStorage 1850
TelSmsSendManualAcknowledge 1852
TelSmsSendMessage 1853

78 Telephony Phone Book 1857

Telephony Phone Book Data Structures 1857
TelPhbEntryType 1858
TelPhbGetAvailablePhonebooksType 1859
TelPhbGetEntriesType 1860
TelPhbGetEntryCountType 1861
TelPhbGetEntryMaxSizesType 1861
Telephony Phone Book Constants 1861
Phone Book Type Constants 1862
Telephony Phone Book Functions 1862
TelPhbAddEntry 1863
TelPhbDeleteEntry 1864
TelPhbGetAvailablePhonebooks 1865
TelPhbGetEntries. 1867
TelPhbGetEntry 1869
TelPhbGetEntryCount 1871
TelPhbGetEntryMaxSizes 1872

TelPhbGetSelectedPhonebook 1874
TelPhbSelectPhonebook 1875

Part IV: Libraries

79 Internet Library **1881**
Internet Library Data Structures 1881
INetCompressionTypeEnum 1881
INetConfigNameType 1882
INetContentTypeEnum 1883
INetHTTPAttrEnum 1883
INetSchemeEnum 1885
INetSettingEnum 1887
INetSockSettingEnum 1888
INetStatusEnum 1890
Internet Library Constants 1892
Configuration Aliases 1892
URL Info Constants 1893
URL Open Constants 1893
Internet Library Functions 1894
INetLibCacheGetObject 1894
INetLibCacheList. 1896
INetLibCheckAntennaState 1898
INetLibClose 1899
INetLibConfigAliasGet 1899
INetLibConfigAliasSet 1900
INetLibConfigDelete 1901
INetLibConfigIndexFromName 1902
INetLibConfigList 1903
INetLibConfigMakeActive. 1904
INetLibConfigRename 1904
INetLibConfigSaveAs. 1905
INetLibGetEvent 1906
INetLibOpen 1907

INetLibSettingGet 1908
INetLibSettingSet 1909
INetLibSockClose 1910
INetLibSockConnect 1911
INetLibSockHTTPAttrGet 1912
INetLibSockHTTPAttrSet 1913
INetLibSockHTTPReqCreate. 1914
INetLibSockHTTPReqSend 1915
INetLibSockOpen 1916
INetLibSockRead. 1918
INetLibSockSettingGet 1919
INetLibSockSettingSet 1920
INetLibSockStatus 1921
IINetLibURLCrack 1922
INetLibURLGetInfo 1923
INetLibURLOpen 1924
INetLibURLsAdd 1926
INetLibWiCmd 1926

80 PalmOSGlue Library 1929

PalmOSGlue Functions 1929
BmpGlueGetBits 1932
BmpGlueGetCompressionType 1933
BmpGlueGetTransparentValue 1933
BmpGlueSetTransparentValue 1934
CtlGlueGetControlStyle 1934
CtlGlueGetFont 1935
CtlGlueGetGraphics 1935
CtlGlueIsGraphical 1936
CtlGlueNewSliderControl 1936
CtlGlueSetFont 1938
CtlGlueSetFrameStyle. 1938
CtlGlueSetLeftAnchor 1939
FldGlueGetLineInfo 1939
FntGlueGetDefaultFontID 1940

FrmGlueGetActiveField 1941
FrmGlueGetDefaultButtonID 1942
FrmGlueGetHelpID 1942
FrmGlueGetLabelFont 1942
FrmGlueGetMenuBarID. 1943
FrmGlueGetObjIDFromObjPtr 1943
FrmGlueGetObjectUsable 1944
FrmGlueNavDrawFocusRing 1944
FrmGlueNavGetFocusRingInfo 1946
FrmGlueNavIsSupported 1947
FrmGlueNavObjectTakeFocus 1948
FrmGlueNavRemoveFocusRing 1949
FrmGlueSetDefaultButtonID. 1949
FrmGlueSetHelpID 1950
FrmGlueSetLabelFont. 1950
IntlGlueGetRoutineAddress 1951
LmGlueGetSystemLocale 1952
LstGlueGetDrawFunction 1952
LstGlueGetFont 1953
LstGlueGetItemsText 1953
LstGlueSetFont 1954
LstGlueSetIncrementalSearch 1954
MemGluePtrNew 1955
SysGlueTrapExists 1955
TblGlueGetColumnMasked 1955
TblGlueGetItemPtr 1956
TxtGlueConvertEncoding 1957
TxtGlueCharIsVirtual 1965
TxtGlueFindString 1966
TxtGlueGetHorizEllipsisChar 1967
TxtGlueGetNumericSpaceChar 1967
TxtGlueLowerChar 1968
TxtGlueLowerStr. 1968
TxtGluePrepFindString 1969
TxtGlueStripSpaces. 1970

TxtGlueTruncateString 1971
TxtGlueUpperChar 1972
TxtGlueUpperStr 1972
UIColorGlueGetNumTableEntries 1973
WinGlueGetFrameType 1973
WinGlueSetFrameType 1974

81 Bluetooth Library: General Functions 1975

Security Functions 1975
BtLibSecurityFindTrustedDeviceRecord 1976
BtLibSecurityGetTrustedDeviceRecordInfo 1977
BtLibSecurityNumTrustedDeviceRecords 1978
BtLibSecurityRemoveTrustedDevice Record 1979
Utility Functions 1979
BtLibAddrAToBtd 1979
BtLibAddrBtdToA 1980
BtLibL2CapHToNL 1981
BtLibL2CapHToNS 1981
BtLibL2CapNToHL 1981
BtLibL2CapNToHS 1982
BtLibRfCommHToNL 1982
BtLibRfCommHToNS 1983
BtLibRfCommNToHL 1983
BtLibRfCommNToHS 1983
BtLibSdpHToNL 1984
BtLibSdpHToNS 1984
BtLibSdpNToHL 1985
BtLibSdpNToHS 1985

82 Bluetooth Library: Management 1987

Bluetooth Management Data Structures 1988
BtLibAccessibleModeEnum 1988
BtLibClassOfDeviceType 1988
BtLibConnectionRoleEnum 1995
BtLibDeviceAddressType 1995
BtLibFriendlyNameType 1996

BtLibManagementEventType 1996
Management Callback Events 1997
btLibManagementEventAccessibilityChange 1997
btLibManagementEventAclConnectInbound 1998
btLibManagementEventAclConnectOutbound 1998
btLibManagementEventAclDisconnect 1999
btLibManagementEventAuthentication Complete 1999
btLibManagementEventEncryptionChange 1999
btLibManagementEventInquiryCanceled 2000
btLibManagementEventInquiryComplete 2000
btLibManagementEventInquiryResult 2000
btLibManagementEventLocalNameChange 2001
btLibManagementEventModeChange 2001
btLibManagementEventNameResult 2002
btLibManagementEventPairingComplete 2003
btLibManagementEventPasskeyRequest 2003
btLibManagementEventPasskeyRequestComplete 2004
btLibManagementEventPiconetCreated 2004
btLibManagementEventPiconetDestroyed 2004
btLibManagementEventRadioState 2004
btLibManagementEventRoleChange 2005
Management Event Status Codes 2006
Library Management Functions. 2008
BtLibClose 2009
BtLibOpen 2009
Management Functions 2011
BtLibCancelInquiry. 2011
BtLibDiscoverMultipleDevices 2012
BtLibDiscoverSingleDevice 2014
BtLibGetGeneralPreference 2016
BtLibGetRemoteDeviceName 2018
BtLibGetSelectedDevices 2020
BtLibLinkConnect 2020
BtLibLinkDisconnect 2022

BtLibLinkGetState 2023
BtLibLinkSetState 2024
BtLibPiconetCreate 2026
BtLibPiconetDestroy 2027
BtLibPiconetLockInbound 2028
BtLibPiconetUnlockInbound 2029
BtLibRegisterManagementNotification 2030
BtLibSetGeneralPreference 2031
BtLibStartInquiry. 2032
BtLibUnregisterManagementNotification 2034
Application-Defined Functions 2034
BtLibManagementCallback 2035

83 Bluetooth Library: Sockets and Service Discovery 2037
Socket-Related Data Structures 2038
BtLibL2CapPsmType 2038
BtLibLanguageBaseTripletType 2038
BtLibProfileDescriptorListEntryType 2053
BtLibProtocolDescriptorListEntryType 2053
BtLibRfCommServerIdType 2054
BtLibSdpAttributeDataType 2054
BtLibSdpAttributeIdType 2055
BtLibSdpRecordHandle 2056
BtLibSdpRemoteServiceRecordHandle 2056
BtLibSdpUuidSizeEnum 2056
BtLibSdpUuidType 2057
BtLibSocketEventType 2059
BtLibSocketRef 2060
BtLibStringType 2060
BtLibUrlType 2060
Socket Callback Events 2061
btLibSocketEventConnectedInbound 2061
btLibSocketEventConnectedOutbound 2062
btLibSocketEventConnectRequest 2062
btLibSocketEventData 2062

btLibSocketEventDisconnected 2063
btLibSocketEventSdpServiceRecordHandle 2063
btLibSocketEventSdpGetAttribute 2064
btLibSocketEventSdpGetStringLen 2065
btLibSocketEventSdpGetNumListEntries 2066
btLibSocketEventSdpGetNumLists 2067
btLibSocketEventSdpGetRawAttribute 2068
btLibSocketEventSdpGetRawAttributeSize 2069
btLibSocketEventSdpGetServerChannelBy Uuid 2070
btLibSocketEventSdpGetPsmByUuid 2070
btLibSocketEventSendComplete 2071
Socket Disconnection Error Codes 2072
Socket Functions 2073
BtLibSocketAdvanceCredit 2073
BtLibSocketClose. 2074
BtLibSocketConnect 2075
BtLibSocketCreate 2078
BtLibSocketGetInfo. 2079
BtLibSocketListen 2082
BtLibSocketRespondToConnection 2085
BtLibSocketSend 2087
Service Discovery Protocol Functions 2088
BtLibSdpCompareUuids 2088
BtLibSdpGetPSMByUuid 2089
BtLibSdpGetRawDataElementSize 2090
BtLibSdpGetRawDataElementType 2091
BtLibSdpGetServerChannelByUuid. 2093
BtLibSdpParseRawDataElement 2094
BtLibSdpServiceRecordCreate 2095
BtLibSdpServiceRecordDestroy 2096
BtLibSdpServiceRecordGetAttribute 2097
BtLibSdpServiceRecordGetNumListEntries 2099
BtLibSdpServiceRecordGetNumLists 2101
BtLibSdpServiceRecordGetRawAttribute 2103

BtLibSdpServiceRecordGetSizeOfRawAttribute 2105
BtLibSdpServiceRecordGetStringOrUrlLength 2106
BtLibSdpServiceRecordMapRemote 2108
BtLibSdpServiceRecordSetAttribute 2109
BtLibSdpServiceRecordSetAttributesForSocket 2111
BtLibSdpServiceRecordSetRawAttribute 2112
BtLibSdpServiceRecordsGetByServiceClass 2114
BtLibSdpServiceRecordStartAdvertising 2116
BtLibSdpServiceRecordStopAdvertising 2117
BtLibSdpUuidInitialize 2118
BtLibSdpVerifyRawDataElement 2118
Application-Defined Functions 2119
BtLibSocketCallback 2120

84 Cryptography Provider Manager 2121

The Default Provider 2121
Fundamental CPM Functions 2122
Using the Crypto-Info Structures 2122
Using the Export Functions 2123
CPM and AP Constants 2124
AP Capability Constants 2124
Block Encryption Mode Constants 2125
Cipher Algorithm Constants 2126
Export Encoding Constants 2128
Hashing Algorithm Constants 2129
Key Class Constants 2130
Key Usage Constants 2130
Plaintext Padding Constants 2131
CPM Error Codes 2132
CPM and AP Structures and Data Types 2134
APCipherInfoStruct 2134
APHashInfoStruct 2135
APKeyInfoStruct 2136
APProviderContextStruct 2137
APProviderInfoStruct 2137

APVerifyInfoStruct 2138
CPMInfoStruct 2139
CPM Functions 2140
CPMLibDecrypt 2140
CPMLibDecryptFinal 2141
CPMLibDecryptInit 2143
CPMLibDecryptUpdate 2144
CPMLibEncrypt 2144
CPMLibEncryptFinal 2146
CPMLibEncryptInit. 2147
CPMLibEncryptUpdate 2148
CPMLibExportCipherInfo 2149
CPMLibExportHashInfo 2150
CPMLibExportKeyInfo 2151
CPMLibExportVerifyInfo 2152
CPMLibGenerateKey 2153
CPMLibGetInfo 2154
CPMLibGetProviderInfo 2154
CPMLibHash 2155
CPMLibHashFinal 2156
CPMLibHashInit 2157
CPMLibHashUpdate 2158
CPMLibImportCipherInfo 2159
CPMLibImportHashInfo 2160
CPMLibImportKeyInfo 2161
CPMLibImportVerifyInfo 2162
CPMLibReleaseCipherInfo 2162
CPMLibReleaseHashInfo 2163
CPMLibReleaseKeyInfo 2163
CPMLibReleaseVerifyInfo 2164
CPMLibVerify 2164
CPMLibVerifyFinal 2166
CPMLibVerifyInit 2167
CPMLibVerifyUpdate 2168

85 SMS Exchange Library **2171**
SMS Exchange Library Data Structures 2171
SmsParamsType 2171
SmsPrefType 2173
SmsReceiveCDMAParamsType 2174
SmsReceiveGSMParamsType 2176
SmsReceiveParamsType 2176
SmsReceiveTDMAParamsType 2178
SmsReportParamsType 2178
SmsSendCDMAParamsType 2179
SmsSendGSMParamsType 2180
SMSSendParamsType 2180
SmsSendTDMAParamsType 2182
SMS Exchange Library Constants 2182
SMS Control Constants 2182
SMS Data Coding Scheme Constants 2184
SMS Network Type Constants 2184
SMS Message Type Constants 2184
SMS Converter Constants 2185

86 SSL Library **2189**
SSL Library Structures and Types 2189
SslAttribute 2189
SslCallback 2190
SslCipherSuiteInfo 2191
SslContext 2192
SslExtendedItem 2193
SslExtendedItems 2194
SslIoBuf 2195
SslLib 2196
SslSession 2197
SslSocket 2198
SslVerify 2199
SSL Library Constants 2202
SSL Open Mode Flags 2202

SSL Close Mode Flags 2202
Mode Attribute Values 2203
Protocol Versions 2204
Compatibility Flags 2204
SSL Callback Commands 2205
Cipher Suite Info Constants 2206
Cipher Suites 2207
Info Callbacks 2207
InfoInterest Values 2209
LastApi Attribute Values 2209
LastIO Attribute Values 2210
SSL Protocol States 2210
SSL Server Alerts 2212
Verification States 2213
SSL Library Errors 2214
SSL Library Traps 2218
Miscellaneous SSL Library Constants 2219
SSL Library Functions and Macros 2219
SslClose 2219
SslConsume 2220
SslContextCreate 2221
SslContextDestroy 2222
SslContextGetLong 2223
SslContextGetPtr 2224
SslContextSetLong 2225
SslContextSetPtr 2226
SslFlush 2227
SslLibClose 2228
SslLibCreate 2229
SslLibDestroy 2229
SslLibGetLong 2230
SslLibGetPtr 2230
SslLibName 2231
SslLibOpen 2232
SslLibSetLong 2232

SslLibSetPtr 2233
SslLibSleep 2234
SslLibWake 2234
SslOpen 2235
SslPeek 2236
SslRead 2237
SslReceive 2238
SslSend 2239
SslWrite 2240
Application-Defined Functions 2241
SslCallbackFunc 2241

87 SSL Library Macros 2251

SSL Library Macro Constants 2251
Attribute Values 2251
SSL Library Macros 2254
SslContextGet_AppInt32 2254
SslContextGet_AppPtr 2254
SslContextGet_AutoFlush 2255
SslContextGet_BufferedReuse 2256
SslContextGet_CipherSuite 2256
SslContextGet_CipherSuiteInfo 2257
SslContextGet_CipherSuites 2258
SslContextGet_ClientCertRequest 2258
SslContextGet_Compat 2259
SslContextGet_DontSendShutdown 2260
SslContextGet_DontWaitForShutdown 2260
SslContextGet_Error 2261
SslContextGet_HsState 2262
SslContextGet_InfoCallback 2262
SslContextGet_InfoInterest 2263
SslContextGet_IoFlags 2263
SslContextGet_IoStruct 2264
SslContextGet_IoTimeout 2265
SslContextGet_LastAlert 2265

SslContextGet_LastApi 2266
SslContextGet_LastIo 2266
SslContextGet_Mode 2267
SslContextGet_PeerCert 2267
SslContextGet_PeerCommonName 2268
SslContextGet_ProtocolVersion 2269
SslContextGet_RbufSize 2270
SslContextGet_ReadBufPending 2270
SslContextGet_ReadOutstanding 2271
SslContextGet_ReadRecPending 2271
SslContextGet_ReadStreaming 2272
SslContextGet_SessionReused 2272
SslContextGet_Socket 2273
SslContextGet_SslSession 2273
SslContextGet_SslVerify 2274
SslContextGet_Streaming 2275
SslContextGet_VerifyCallback 2275
SslContextGet_WbufSize 2276
SslContextGet_WriteBufPending 2276
SslContextSet_AppInt32 2277
SslContextSet_AppPtr 2278
SslContextSet_AutoFlush 2278
SslContextSet_BufferedReuse 2279
SslContextSet_CipherSuites 2280
SslContextSet_Compat 2280
SslContextSet_DontSendShutdown 2281
SslContextSet_DontWaitForShutdown 2282
SslContextSet_Error 2282
SslContextSet_InfoCallback 2283
SslContextSet_InfoInterest 2284
SslContextSet_IoFlags 2285
SslContextSet_IoStruct 2285
SslContextSet_IoTimeout 2286
SslContextSet_LastAlert 2287
SslContextSet_Mode 2287

SslContextSet_ProtocolVersion 2288
SslContextSet_RbufSize 2289
SslContextSet_ReadStreaming 2289
SslContextSet_Socket 2290
SslContextSet_SslSession 2291
SslContextSet_VerifyCallback 2291
SslContextSet_WbufSize 2292
SslLibGet_AppInt32 2293
SslLibGet_AppPtr 2293
SslLibGet_AutoFlush 2294
SslLibGet_BufferedReuse 2295
SslLibGet_CipherSuites 2295
SslLibGet_Compat 2296
SslLibGet_DontSendShutdown 2297
SslLibGet_DontWaitForShutdown 2297
SslLibGet_InfoCallback 2298
SslLibGet_InfoInterest 2299
SslLibGet_Mode 2299
SslLibGet_ProtocolVersion 2300
SslLibGet_RbufSize 2300
SslLibGet_ReadStreaming 2301
SslLibGet_VerifyCallback 2301
SslLibGet_WbufSize 2302
SslLibSet_AppInt32 2303
SslLibSet_AppPtr 2303
SslLibSet_AutoFlush 2304
SslLibSet_BufferedReuse 2305
SslLibSet_CipherSuites 2305
SslLibSet_Compat 2306
SslLibSet_DontSendShutdown 2307
SslLibSet_DontWaitForShutdown 2307
SslLibSet_InfoCallback 2308
SslLibSet_InfoInterest 2309
SslLibSet_Mode 2310
SslLibSet_ProtocolVersion 2310

SslLibSet_RbufSize 2311
SslLibSet_ReadStreaming 2312
SslLibSet_VerifyCallback 2312
SslLibSet_WbufSize 2313

88 Personal Data Interchange Library 2315

PDI Library Data Structures 2315
PdiDictionary 2315
PdiReaderType 2316
PdiWriterType 2318
PDI Library Constants. 2318
Buffer Management Constants 2319
Encoding Type Constants 2320
Error Code Constants 2320
Parameter Name Constants 2321
Parameter Value Constants 2322
Property Name Constants 2323
Property Type Constants 2323
Property Value Field Constants 2324
Property Value Format Constants 2325
Reader and Writer Options Constants 2326
Reader Event Constants 2329
Value Type Constants 2330
PDI Library Functions. 2331
PdiDefineReaderDictionary 2331
PdiDefineResizing 2332
PdiDefineWriterDictionary 2332
PdiEnterObject. 2334
PdiLibClose 2334
PdiLibOpen 2335
PdiParameterPairTest 2335
PdiReaderDelete 2336
PdiReaderNew. 2337
PdiReadParameter 2338
PdiReadProperty. 2338

PdiReadPropertyField 2339
PdiReadPropertyName 2341
PdiSetCharset 2342
PdiSetEncoding 2343
PdiWriteBeginObject 2343
PdiWriteEndObject 2344
PdiWriteParameter 2345
PdiWriteParameterStr 2347
PdiWriteProperty 2348
PdiWritePropertyBinaryValue 2348
PdiWritePropertyFields 2349
PdiWritePropertyStr 2350
PdiWritePropertyValue 2351
PdiWriterDelete 2352
PdiWriterNew 2353

89 Unified Data Access Manager 2355

UDA Manager Data Structures 2356
UDABufferSize 2356
UDAObjectType 2356
UDAFilterType 2356
UDAReaderType 2357
UDAWriterType 2358
UDA Manager Constants 2358
Control Flags 2359
Error Constants 2359
Object Option Flags 2359
Miscellaneous Constants 2360
UDA Manager Functions 2360
UDAControl 2360
UDADelete 2361
UDAEndOfReader 2362
UDAFilterJoin 2362
UDAInitiateWrite 2363
UDAMoreData 2363

UDARead 2364
UDAWriterFlush 2364
UDAWriterJoin 2365
UDA Object Creation Functions 2366
UDAExchangeReaderNew 2366
UDAExchangeWriterNew 2366
UDAMemoryReaderNew 2367

Part V: Appendixes

A System Use Only Functions **2371**

B Compatibility Guide **2381**

2.0 New Feature Set 2382
3.0 New Feature Set 2385
3.1 New Feature Set 2389
3.2 New Feature Set 2392
International Feature Set 2394
Japanese Feature Set 2396
Wireless Internet Feature Set 2396
New Serial Manager Feature Set 2398
Connection Manager Feature Set 2401
3.5 New Feature Set 2402
Notification Feature Set 2408
4.0 New Feature Set 2409
Expansion Manager Feature Set 2415
VFS Manager Feature Set 2416
Bluetooth Library Feature Set 2418
High-Density Display Feature Set 2421
1.5x Display Feature Set 2423
Sound Stream Feature Set 2424
5.0 New Feature Set 2426
5.1 New Feature Set 2431
Pen Input Manager Feature Set 2434

Graffiti 2 Feature Set 2436
5.3SC New Feature Set 2437
5.4 New Feature Set 2438
Bluetooth Sleep and Wake Scheduling Feature Set. 2439
ARM Debug Nub Feature Set. 2440
Palm OS Cobalt Feature Set 2440
One-Handed Navigation Feature Set 2459

C 1.0 Float Manager 2461

Float Manager Functions 2461
FplAdd 2461
FplAToF. 2462
FplBase10Info 2462
FplDiv 2463
FplFloatToLong 2463
FplFloatToULong 2464
FplFree 2464
FplFToA 2464
FplInit 2465
FplLongToFloat 2465
FplMul 2466
FplSub 2466

D Palm OS Error Codes 2467

Index 2511

About This Document

Palm OS Programmer's API Reference is part of the Palm OS® Software Development Kit. This introduction provides an overview of SDK documentation, discusses what materials are included in this document, and what conventions are used.

Palm OS SDK Documentation

The following documents are part of the SDK:

Document	Description
Palm OS Programmer's API Reference	An API reference document that contains descriptions of all Palm OS function calls and important data structures.
Palm OS Programmer's Companion	A multi-volume guide to application programming for the Palm OS. This guide contains conceptual and "how-to" information that complements the Reference.
Constructor for Palm OS	A guide to using Constructor to create Palm OS resource files.
Palm OS Programming Development Tools Guide	A guide to writing and debugging Palm OS applications with the various tools available.

What This Volume Contains

This section provides an overview of this volume.

- Part I, "User Interface," documents the API contained in the header files in the `\Incs\Core\UI\` folder. This part contains chapters covering subjects such as application launch codes, user interface resources, events, and all window, form, and field object managers.
- Part II, "System Management," documents the API contained in the header files in the `\Incs\Core\System\` folder. This part contains chapters covering subjects such as the alarm manager, data and resource manager, feature manager, float

manager, graffiti manager, key manager, memory manager, preferences manager, sound manager, string manager, and system manager.

- Part III, "Communications," documents the API related to communications, such as the exchange manager, IR library, net library, Secure Sockets Layer (SSL) library, serial manager, and serial drivers.
- Part IV, "Libraries," documents the API contained in the header files in the `\Incs\Libraries\` folder. This part contains chapters covering the Internet Library and the Palm OS Glue library.

Additional Resources

- Documentation

  PalmSource publishes its latest versions of this and other documents for Palm OS developers at

  http://www.palmos.com/dev/support/docs/

- Training

  PalmSource and its partners host training classes for Palm OS developers. For topics and schedules, check

  http://www.palmos.com/dev/training

- Knowledge Base

  The Knowledge Base is a fast, web-based database of technical information. Search for frequently asked questions (FAQs), sample code, white papers, and the development documentation at

  http://www.palmos.com/dev/support/kb/

Conventions Used in This Guide

This guide uses the following typographical conventions:

This style...	Is used for...
`fixed width font`	Code elements such as function, structure, field, bitfield.
italic	Emphasis (for other elements).
blue and underlined	Hot links.

Part I
User Interface

1

Application Launch Codes

This chapter provides detailed information about the predefined application launch codes. Launch codes are declared in the header file `SystemMgr.h`. The associated parameter blocks are declared in `AppLaunchCmd.h`, `AlarmMgr.h`, `ExgMgr.h`, and `Find.h`.

Table 1.1 lists all Palm OS® standard launch codes. More detailed information is provided immediately after the table:

- Launch Codes
- Launch Flags

To learn what a launch code is and how to use it, see the chapter titled "Application Startup and Stop" in the *Palm OS Programmer's Companion*, vol. I.

Table 1.1 Palm OS Launch Codes

Code	Request
`scptLaunchCmdExecuteCmd`	Execute the specified Network login script plugin command.
`scptLaunchCmdListCmds`	Provide information about the commands that your Network script plugin executes.
`sysAppLaunchCmdAddRecord`	Add a record to a database.
`sysAppLaunchCmdAlarmTriggered`	Schedule next alarm or perform quick actions such as sounding alarm tones.
`sysAppLaunchCmdAttention`	Perform the action requested by the attention manager.

Table 1.1 Palm OS Launch Codes *(continued)*

Code	Request
sysAppLaunchCmdCardLaunch	Launch the application. This launch code signifies that the application is being launched from an expansion card.
sysAppLaunchCmdCountryChange	Respond to country change.
sysAppLaunchCmdDisplayAlarm	Display specified alarm dialog or perform time-consuming alarm-related actions.
sysAppLaunchCmdExgAskUser	Let application override display of dialog asking user if they want to receive incoming data via the Exchange Manager.
sysAppLaunchCmdExgGetData	Notify application that it should send data using the Exchange Manager.
sysAppLaunchCmdExgPreview	Notify application that it should display a preview using the Exchange Manager.
sysAppLaunchCmdExgReceiveData	Notify application that it should receive incoming data using the Exchange Manager.
sysAppLaunchCmdFind	Find a text string.
sysAppLaunchCmdGoTo	Go to a particular record, display it, and optionally select the specified text.
sysAppLaunchCmdGoToURL	Launch an application and open a URL.
sysAppLaunchCmdHandleSyncCallApp	Perform some application-specific operation at the behest of the application's conduit.
sysAppLaunchCmdInitDatabase	Initialize database.

Table 1.1 Palm OS Launch Codes *(continued)*

Code	Request
sysAppLaunchCmdLookup	Look up data. In contrast to sysAppLaunchCmdFind, a level of indirection is implied. For example, look up a phone number associated with a name.
sysAppLaunchCmdNormalLaunch	Launch normally.
sysAppLaunchCmdNotify	Notify about an event.
sysAppLaunchCmdOpenDB	Launch application and open a database.
sysAppLaunchCmdPanelCalledFromApp	Tell preferences panel that it was invoked from an application, not the Preferences application.
sysAppLaunchCmdReturnFromPanel	Tell an application that it's restarting after preferences panel had been called.
sysAppLaunchCmdSaveData	Save data. Often sent before find operations.
sysAppLaunchCmdSyncNotify	Notify applications that a HotSync® has been completed.
sysAppLaunchCmdSystemLock	Sent to the Security application to request that the system be locked down.
sysAppLaunchCmdSystemReset	Respond to system reset. No UI is allowed during this launch code.
sysAppLaunchCmdTimeChange	Respond to system time change.
sysAppLaunchCmdURLParams	Launch an application with parameters from the Web Clipping Application Viewer.

Launch Codes

This section provides supplemental information about launch codes. For some launch codes, it lists the parameter block, which in some cases provides additional information about the launch code.

sysAppLaunchCmdAddRecord

Add a record to an application's database.

This launch code is used to add a message to the Mail or iMessenger™ (on the Palm VII™ organizer) application's outbox. You pass information about the message such as address, body text, etc. in the parameter block. For iMessenger, you can set the `edit` field of the parameter block to control whether or not the iMessenger editor is displayed. Set it to `true` to display the editor or `false` not to display it.

For more information on sending messages via iMessenger, see "Sending Email Messages" on page 234 in the *Palm OS Programmer's Companion*, vol. II, *Communications*.

IMPORTANT: Implemented for iMessenger only if Wireless Internet Feature Set is present. Implemented for Mail only on OS version 3.0 or later.

sysAppLaunchCmdAddRecord Parameter Block for Mail Application

Prototype

```
typedef enum {
   mailPriorityHigh,
   mailPriorityNormal,
   mailPriorityLow
} MailMsgPriorityType;

typedef struct {
   Boolean secret;
   Boolean signature;
   Boolean confirmRead;
   Boolean confirmDelivery;
   MailMsgPriorityType priority;
```

```
   UInt8 padding
   Char *subject;
   Char *from;
   Char *to;
   Char *cc;
   Char *bcc;
   Char *replyTo;
   Char *body;
} MailAddRecordParamsType;
```

Fields

`secret`
: `True` means that the message should be marked secret.

`signature`
: `True` means that the signature from the Mail application's preferences should be attached to the message.

`confirmRead`
: `True` means that a confirmation should be sent when the message is read.

`confirmDelivery`
: `True` means that a confirmation should be sent when the message is delivered.

`priority`
: Message priority. Specify one of the `MailMsgPriorityType` enumerated types.

`padding`
: Reserved for future use.

`subject`
: Message's subject, a null-terminated string (optional).

`from`
: Message's sender, a null-terminated string (not used on outgoing mail).

`to`
: Address of the recipient, a null-terminated string (required).

`cc`
: Addresses of recipients to be copied, a null-terminated string (optional).

`bcc`
: Addresses of recipients to be blind copied, a null-terminated string (optional).

`replyTo`
: Reply to address, a null-terminated string (optional).

`body`
: The text of the message, a null-terminated string (required).

sysAppLaunchCmdAddRecord Parameter Block for iMessenger Application

Prototype

```
typedef struct {
   UInt16 category;
   Boolean edit;
   Boolean signature;
   Char *subject;
   Char *from;
   Char *to;
   Char *replyTo;
   Char *body;
} MsgAddRecordParamsType;
```

Fields

`category`
: Category in which to place the message. Specify one of the following categories:

  `MsgInboxCategory`

  `MsgOutboxCategory`

  `MsgDeletedCategory`

  `MsgFiledCategory`

  `MsgDraftCategory`

`edit`
: `True` means that the message should be opened in the editor. `False` means that the message should simply be placed into the outbox and the editor not opened. You can specify `true` only if the category is set to `MsgOutboxCategory`.

`signature`
: `True` means that the signature from the iMessenger application preferences should be attached to the message.

`subject`
: Message's subject, a null-terminated string (optional).

`from`
: Message's sender, a null-terminated string (not used on outgoing mail).

`to`
: Address of the recipient, a null-terminated string (required).

`replyTo`
: Reply to address, a null-terminated string (optional).

`body`
: The text of the message, a null-terminated string (required).

sysAppLaunchCmdAlarmTriggered

Performs quick action such as scheduling next alarm or sounding alarm.

This launch code is sent as close to the actual alarm time as possible. An application may perform any quick, non-blocking action at this time. Multiple alarms may be pending at the same time for multiple applications, and one alarm display shouldn't block the system and prevent other applications from receiving their alarms in a timely fashion. An opportunity to perform more time-consuming actions will come when `sysAppLaunchCmdDisplayAlarm` is sent.

sysAppLaunchCmdAlarmTriggered Parameter Block

Prototype

```
typedef struct SysAlarmTriggeredParamType {
   UInt32 ref;
   UInt32 alarmSeconds;
   Boolean purgeAlarm;
   UInt8 padding;
} SysAlarmTriggeredParamType;
```

Fields

→ *ref*
: The caller-defined value specified when the alarm was set with `AlmSetAlarm()`.

→ *alarmSeconds*
The date/time specified when the alarm was set with `AlmSetAlarm`. The value is given as the number of seconds since 1/1/1904.

← *purgeAlarm*
Upon return, set to true if the alarm should be removed from the alarm table. Use this as an optimization to prevent the application from receiving `sysAppLaunchCmdDisplayAlarm` if you don't wish to perform any other processing for this alarm. If you do want to receive the launch code, set this field to false.

`padding`
Not used.

sysAppLaunchCmdAttention

Purpose Perform the action requested by the attention manager. This launch code is accompanied by a value of the `AttnCommand` type; this type specifies the set of possible commands that can be sent to the application that requested the alarm.

Prototype

```
typedef UInt16 AttnCommand;
```

Comments The following table lists the values that `AttnCommand` can assume.

Table 1.2 sysAppLaunchCmdAttention Commands

Constant	Value	Description
`AttnCommand_drawDetail`	`((AttnCommand)1)`	Indicates that the application needs to draw the detailed contents of the attention dialog. The command arguments parameter points to a structure of type `AttnCommandDrawDetailArgsType`.
`AttnCommand_drawList`	`((AttnCommand)2)`	Indicates that the application needs to draw the appropriate list item in the attention dialog. The command arguments parameter points to a structure of type `AttnCommandDrawListArgsType`.
`AttnCommand_customEffect`	`((AttnCommand)3)`	Indicates that the Attention Manager is doing something to get the user's attention, and any application-specific special effect should be done. This command is only sent to attention items that set the `AttnFlags_CustomEffectBit` when they call `AttnGetAttention()`, which most applications won't do.
`AttnCommand_goThere`	`((AttnCommand)4)`	Tells the application to navigate to the item. The command arguments parameter is `NULL`. An application commonly calls `SysAppLaunch()` upon receipt of this command to have itself launched.

Table 1.2 sysAppLaunchCmdAttention Commands

Constant	Value	Description
`AttnCommand_gotIt`	`((AttnCommand)5)`	Tells the application that the user is dismissing the item. The command arguments parameter is `NULL`. The application may choose to clean up memory at this point.
`AttnCommand_snooze`	`((AttnCommand)6)`	Indicates to the application that the user is snoozing. The command arguments parameter is `NULL`. Most applications do nothing upon receipt of this command. This command is passed to each and every item currently pending, insistent or subtle. Applications with more than one attention item pending are called more than once.
`AttnCommand_iterate`	`((AttnCommand)7)`	This command is passed to the application during the enumeration of attention items. This command is particularly useful after HotSync operations, as it allows the application to examine each item, updating or removing those that are stale or invalid.

AttnCommandDrawDetailArgsType

Purpose When `AttnCommand_drawDetail` is passed to the application, either via the callback function or as a parameter accompanying the `sysAppLaunchCmdAttention` launch code, the application needs to draw the detailed contents of the attention dialog. The `AttnCommandDrawDetailArgsType` structure accompanies the

`AttnCommand_drawDetail` command, and provides the information needed to draw the contents of that dialog.

Prototype

```
typedef struct {
  RectangleType bounds;
  Boolean firstTime;
  AttnFlagsType flags;
} AttnCommandDrawDetailArgsType;
```

Fields

`bounds`
: Contains the window-relative bounding box for the area to draw. The clipping region is also set to the dimensions of this box to prevent accidental drawing outside.

`firstTime`
: Set to `true` if the user has not yet seen this item. The value of this field could be used, for example, to display attentions that the user hasn't seen before in some special way.

`flags`
: The global user preferences for this attention attempt combined with the custom flags passed in by the developer. For example, if the global preference is to mute sounds, and the developer flags are both zero, then the `AttnFlags_NoSound` flag is on and the `AttnFlags_AlwaysSound` flag is off.

AttnCommandDrawListArgsType

When `AttnCommand_drawList` is passed to the application, either via the callback function or as a parameter accompanying the `sysAppLaunchCmdAttention` launch code, the application is to draw the appropriate list item in the attention dialog. The `AttnCommandDrawListArgsType` structure accompanies the `AttnCommand_drawList` command, and provides the information needed to draw the contents of that dialog.

Prototype

```
typedef struct {
  RectangleType bounds;
  Boolean firstTime;
  AttnFlagsType flags;
} AttnCommandDrawListArgsType;
```

Fields

`bounds`
: Contains the window-relative bounding box for the area to draw. The clipping region is also set to the dimensions of this box to prevent accidental drawing outside.

`firstTime`
: Set to `true` if the user has not yet seen this item. The value of this field could be used, for example, to trigger a custom sound the first time this attention item is presented to the user.

`flags`
: The global user preferences for this attention attempt combined with the custom flags passed in by the developer. For example, if the global preference is to mute sounds, and the developer flags were both zero, then the `AttnFlags_NoSound` flag is on and the `AttnFlags_AlwaysSound` flag is off.

AttnCommandGotItArgsType

When `AttnCommand_gotIt` is passed to the application, either via the callback function or as a parameter accompanying the `sysAppLaunchCmdAttention` launch code, it is accompanied by an `AttnCommandGotItArgsType` structure. This structure indicates whether the `AttnCommand_gotIt` command was generated because the user dismissed the attention, or whether the system is simply informing your application that `AttnForgetIt()` was called. Your application normally ignores the latter case if your application made the call to `AttnForgetIt`.

Prototype

```
typedef struct {
  Boolean dismissedByUser;
} AttnCommandGotItArgsType;
```

Fields

`dismissedByUser`
: `true` indicates that the user dismissed the attention. `false` indicates that the `AttnCommand_gotIt` command was generated by a call to `AttnForgetIt`.

sysAppLaunchCmdCardLaunch

This launch code is sent to applications that are being run from an expansion card. The application is copied into the device's main

memory prior to being sent this launch code. If the application doesn't respond to `sysAppLaunchCmdCardLaunch`, it is then sent a `sysAppLaunchNormalLaunch` launch code. Applications that can profit from the knowledge that they are being launched from an expansion card may want to consult the fields in the parameter block that accompanies `sysAppLaunchCmdCardLaunch`.

When the Launcher sends `sysAppLaunchCmdCardLaunch` to an application, it also sends `sysAppLaunchFlagNewGlobals`, and `sysAppLaunchFlagUIApp` flags. These two flags are not sent to `start.prc`, however. Applications should never interact with the user upon receiving this launch code, and should not depend on globals being available. This launch code is intended to notify the application that it is being launched from a card. Applications typically save some state information upon receiving this launch code and do the bulk of their processing when they receive `sysAppLaunchNormalLaunch`.

sysAppLaunchCmdCardLaunch Parameter Block

Prototype

```
typedef struct {
   Err err;
   UInt16 volRefNum;
   const Char *path;
   UInt16 startFlags;
} SysAppLaunchCmdCardType;
```

Fields

← *err*
: Initially set to `expErrUnsupportedOperation`, applications that recognize `sysAppLaunchCmdCardLaunch` and that don't want to receive the subsequent `sysAppLaunchNormalLaunch` launch code should set this field to `errNone`.

→ *volRefNum*
: The reference number of the volume from which the application is being launched.

→ *path*
: The complete path to the application being launched.

↔ *startFlags*
: This field is made up of a combination of the following flags.

`sysAppLaunchStartFlagAutoStart`
: Indicates that the application is being run automatically upon card insertion.

`sysAppLaunchStartFlagNoUISwitch`
: Set this bit to prevent a UI switch to the application.

`sysAppLaunchStartFlagNoAutoDelete`
: Set this bit to prevent the VFS Manager from deleting the copy of the application in main memory when the associated volume is unmounted.

sysAppLaunchCmdCountryChange

Responds to country change.

Applications should change the display of numbers to use the proper number separators. To do this, call `LocGetNumberSeparators()`, `StrLocalizeNumber`, and `StrDelocalizeNumber`.

sysAppLaunchCmdDisplayAlarm

Performs full, possibly blocking, handling of alarm.

This is the application's opportunity to handle an alarm in a lengthy or blocking fashion. Alert dialogs are usually displayed when this launch code is received. This work should be done here, not when `sysAppLaunchCmdAlarmTriggered` is received. Multiple alarms may be pending at the same time for multiple applications, and one alarm display shouldn't block the system and prevent other applications from receiving their alarms in a timely fashion.

sysAppLaunchCmdDisplayAlarm Parameter Block

Prototype

```
typedef struct SysDisplayAlarmParamType {
   UInt32 ref;
   UInt32 alarmSeconds;
   Boolean soundAlarm;
   UInt8 padding;
} SysDisplayAlarmParamType;
```

Fields

→ *ref*
The caller-defined value specified when the alarm was set with `AlmSetAlarm()`.

→ *alarmSeconds*
The date/time specified when the alarm was set with `AlmSetAlarm`. The value is given as the number of seconds since 1/1/1904.

→ *soundAlarm*
`true` if the alarm should be sounded, `false` otherwise. This value is currently not used.

`padding`
Not used.

sysAppLaunchCmdExgAskUser

The Exchange Manager sends the `sysAppLaunchCmdExgAskUser` launch code to the application when data has arrived for that application. This launch code allows the application to tell the Exchange Manager not to display the exchange dialog, which it uses to have the user confirm the receipt of data. If the application does not handle this launch code, the default course of action is that the Exchange Manager displays the exchange dialog.

Applications may want to respond to this launch code under these circumstances:

- To reject all incoming data or to reject data under certain circumstances without first prompting the user. To reject incoming data, set the `result` field of the parameter block to `exgAskCancel` and then return.
- To receive incoming data without confirmation. To automatically receive incoming data, set the result field to `exgAskOk`.
- To provide a user confirmation dialog with extra functionality. This is described in more detail below.

Starting with Palm OS 3.5, the Exchange Manager allows applications to provide extra functionality in the exchange dialog. You can have the dialog include a category pop-up list from which the user chooses a category in which to file the incoming data. If you

want to provide a category pop-up list, call the `ExgDoDialog()` function in response to this launch code and pass it a database that contains the categories to be listed. See the description of that function for more information.

Applications may also bypass the call to `ExgDoDialog` altogether and provide their own dialogs.

If an application responds to this launch code, it must set the `result` field in the parameter block to the appropriate value. Possible values are:

`exgAskDialog`
: Display the default exchange dialog provided by Exchange Manager.

`exgAskOk`
: Accept the incoming data.

`exgAskCancel`
: Reject the incoming data.

On Palm OS 3.5 or higher if you don't use the default version of the dialog, return `exgAskOk` if the user confirmed or `exgAskCancel` if the user canceled. If you don't set the `result` field properly, two dialogs are displayed.

IMPORTANT: Implemented only if 3.0 New Feature Set is present.

---

sysAppLaunchCmdExgAskUser Parameter Block

Prototype

```
typedef struct {
   ExgSocketPtr socketP;
   ExgAskResultType result;
   UInt8 reserved;
} ExgAskParamType;
```

Fields

↔ *socketP*
: Socket pointer (see `ExgSocketType`)

← *result*
: Show dialog, auto-confirm, or auto-cancel

→ *reserved*
: Reserved for future use

sysAppLaunchCmdExgGetData

The Exchange Manager sends the `sysAppLaunchCmdExgGetData` launch code when the exchange library requests data to be sent to a remote device. That is, an application on a remote device has performed an `ExgGet()` function to request data, and the Exchange Manager has determined that the launched application should handle the request.

To respond to this launch code, applications should initiate a connection with `ExgPut()`, use `ExgSend()` to send the data, and call `ExgDisconnect()` when finished.

The parameter block sent with this launch code is a pointer to the `ExgSocketType` structure corresponding to the Exchange Manager connection on which the data is to be sent. You pass this socket pointer to `ExgPut`. For more details, see the "Exchange Manager" chapter.

IMPORTANT: Implemented only if 4.0 New Feature Set is present.

sysAppLaunchCmdExgPreview

Following the launch code `sysAppLaunchCmdExgAskUser`, the Exchange Manager sends the `sysAppLaunchCmdExgPreview` launch code to have the application display the preview in the exchange dialog.

sysAppLaunchCmdExgPreview Parameter Block

Prototype

```
typedef struct {
   UInt16 version;
   ExgSocketType *socketP;
   UInt16 op;
   Char *string;
   UInt32 size;
   RectangleType bounds;
   UInt16 types;
   Err error;
} ExgPreviewInfoType;
```

Fields

→ *version*
Set this field to 0 to specify version 0 of this structure.

→ *socketP*
A pointer to the socket structure (see `ExgSocketType`). The `libraryRef` field must point to the exchange library from which preview data should be received.

→ *op*
A constant that identifies the operation. This can be one of the following:

`exgPreviewDialog`
Display a form or modal dialog containing the preview. This constant is only used in situations where one application launches another to display data.

`exgPreviewDraw`
Draw the preview as a graphic in the `bounds` rectangle.

`exgPreviewLongString`
Return the preview as a long string in the `string` field.

`exgPreviewQuery`
Return the list of preview modes the application supports in the `types` field.

`exgPreviewLongString`
Return the preview as a short string in the `string` field.

← *string*
A buffer into which the preview string is placed if one of the string preview operations is specified.

→ *size*
The allocated size of the `string` field.

→ *bounds*
The bounds of the rectangle in which to draw the graphic if the preview operation is `exgPreviewDraw`.

← *types*
Upon return from `exgPreviewQuery`, a bit field identifying the types of previews the library supports

(`exgPreviewDraw`, `exgPreviewLongString`, or `exgPreviewShortString`).

← *error*
The error code returned from the library. If this is `errNone`, the preview operation was successful.

Applications that respond to this launch code should check the parameter block's `op` field and respond as described above.

Applications can define and use their own constants for the preview operation. Operations specific to an application are numbered starting at `exgPreviewFirstUser` and should be no greater than `exgPreviewLastUser`.

Applications respond to this launch in much the same way they respond to `sysAppLaunchCmdExgReceiveData`. Use `ExgAccept()` to accept the preview connection, `ExgReceive()` to receive the data, and then `ExgDisconnect()` to end the connection. The only difference is what is done with the data when it is received. With this launch code, the application should return the data in the `string` field or draw it in the `bounds` rectangle. With the `sysAppLaunchCmdExgReceiveData` launch code, the application stores the received data.

IMPORTANT: Implemented only if 4.0 New Feature Set is present.

sysAppLaunchCmdExgReceiveData

The Exchange Manager sends the sysAppLaunchCmdExgReceiveData launch code following the `sysAppLaunchCmdExgAskUser` and `sysAppLaunchCmdExgPreview` launch codes to notify the application that it should receive the data (assuming that the application and/or the user has indicated the data should be received).

The application should use Exchange Manager functions to receive the data and store it or do whatever it needs to with the data. Specifically, most applications should respond to this launch code

by calling `ExgAccept()` to accept the connection and then `ExgReceive()` to receive the data.

Note that the application may not be the active application, and thus may not have globals available when it is launched with this launch code. You can check if you have globals by using this code in the `PilotMain()` routine:

```
Boolean appIsActive = launchFlags & sysAppLaunchFlagSubCall;
```

The `appIsActive` value is `true` if your application is active and globals are available; otherwise, you won't be able to access any of your global variables during the receive operation.

The parameter block sent with this launch code is a pointer to the `ExgSocketType` structure corresponding to the Exchange Manager connection on which the data is arriving. Pass this pointer to the `ExgAccept` function to begin receiving the data. For more details, refer to the "Exchange Manager" chapter.

IMPORTANT: Implemented only if 3.0 New Feature Set is present.

sysAppLaunchCmdFind

This launch code is used to implement the global find. When the user enters a text string in the Find dialog, the system sends this launch code with the `FindParamsType` parameter block to each application. The application should search for the string that the user entered and return any records matching the find request.

The system displays the results of the query in the Find results dialog. The system continues the search with each application until it has a full screen of matching records or until all of the applications on the device have had a chance to respond. If the screen is full, a Find More button appears at the bottom of the dialog. If the user clicks the Find More button, the search resumes. Applications can test whether the current find is a continuation of a previous one by checking the `continuation` field in the parameter block.

Most applications that use text records should support this launch code. When they receive it, they should search all records for matches to the find string and return all matches. Functions that you can use to respond to this launch code are described in Chapter 10, "Find."

An application can also integrate the find operation in its own user interface and send the launch code to a particular application.

Applications that support this launch code should support `sysAppLaunchCmdSaveData` and `sysAppLaunchCmdGoTo` as well.

sysAppLaunchCmdFind Parameter Block

Prototype

```
typedef struct {
   UInt16 dbAccesMode;
   UInt16 recordNum;
   Boolean more;
   Char strAsTyped [maxFindStrLen+1];
   Char strToFind [maxFindStrLen+1];
   UInt8 reserved1;
   UInt16 numMatches;
   UInt16 lineNumber;
   Boolean continuation;
   Boolean searchedCaller;
   LocalID callerAppDbID;
   UInt16 callerAppCardNo;
   LocalID appDbID;
   UInt16 appCardNo;
   Boolean newSearch;
   UInt8 reserved2;
   DmSearchStateType searchState;
   FindMatchType match [maxFinds];
} FindParamsType;
```

Fields

`dbAccesMode`

Mode in which to open the application's database. Pass this directly to `DmOpenDatabase()` as the `mode` parameter. Its value is either `dmModeReadOnly` or `dmModeReadOnly | dmModeShowSecret`. (See "Open Mode Constants" for more information.)

`recordNum`
: Index of last record that contained a match. Start the search from this location. Do not set this value directly. Instead, call `FindSaveMatch()` when you have a matching record.

`more`
: If `true`, the Find results dialog displays the Find More button indicating that there may be more results to display.

  Typically `FindSaveMatch()` handles setting the `more` field. Applications with large databases to search may want to periodically check for and stop the search if an event is pending. If so, they should set this field to `true` before stopping.

`strAsTyped`
: Search string as the user entered it.

`strToFind`
: Normalized version of the search string. The method by which a search string is normalized varies depending on the version of Palm OS and the character encoding supported by the device. You pass `strToFind` directly to the search function (either `FindStrInStr()`, `TxtFindString()`, or `TxtGlueFindString`).

`reserved1`
: Reserved for future use.

`numMatches`
: The current number of matches. Do not set this field directly. Instead, call `FindSaveMatch()`, which increments it for you.

`lineNumber`
: Line number of the next line that displays the results. Do not set this field directly. It is incremented by a call to `FindDrawHeader()`.

`continuation`
: If `true`, the launch code is being sent as part of a continuation of a previous Find. If `false`, this is a new Find. Do not set this field directly; the system sets it when the Find results dialog is full.

`searchedCaller`
: If `true`, the application that was active at the time the user tapped the Find button has responded to this launch code. This application is always searched before any others.

`callerAppDbID`
: Database ID of the application that was active when the user tapped the Find button. Do not change the value of this field; the system sets it and uses it when searching for application databases.

`callerAppCardNo`
: Card number of the application that was active when the user tapped the Find button. Do not change the value of this field; the system sets it and uses it when searching for application databases.

`appDbID`
: The ID of your application's resource database. Do not set this field directly; the system sets it and uses it when searching for application databases.

`appCardNo`
: The card number of your application's resource database. Do not set this field directly; the system sets it and uses it when searching for application databases.

`newSearch`
: System use only.

`reserved2`
: Reserved for future use.

`searchState`
: System use only.

`match`
: System use only.

sysAppLaunchCmdGoTo

Sent in conjunction with `sysAppLaunchCmdFind` or `sysAppLaunchCmdExgReceiveData` to allow users to actually inspect the record that the global find returned or that was received by the Exchange Manager.

Applications should do most of the normal launch actions, then display the requested item. The application should continue running unless explicitly closed.

An application launched with this code does have access to global variables, static local variables, and code segments other than segment 0 (in multi-segment applications).

Applications that receive this launch code should test the `sysAppLaunchFlagNewGlobals` launch flag to see if they need to initialize global variables. `sysAppLaunchFlagNewGlobals` indicates that the system has just allocated your global variables.

For example:

```
case sysAppLaunchCmdGoTo:
  if (launchFlags & sysAppLaunchFlagNewGlobals)
    StartApplication();
```

Note that you shouldn't automatically initialize the global variables in response to this launch code. Test the launch flag first. Your application receives this launch code when the user selects a record in the global find results. If your application was the current application before the user selected the Find command, the launch flag is clear to indicate that your globals should not be re-initialized.

sysAppLaunchCmdGoTo Parameter Block

Prototype

```
typedef struct {
   Int16 searchStrLen;
   UInt16 dbCardNo;
   LocalID dbID;
   UInt16 recordNum;
   UInt16 matchPos;
   UInt16 matchFieldNum;
   UInt32 matchCustom;
} GoToParamsType;
```

Fields

`searchStrLen`
: Length of normalized search string. This is *not* the length of the matching string. See below for a full explanation.

`dbCardNo`
: Card number of the database to open.

`dbID`
: Local ID of the database to open.

`recordNum`
: Index of record containing a match.

`matchPos`
: Position of the match within the field.

`matchFieldNum`
: Field number string was found in.

`matchCustom`
: Application-specific information.

Often, applications highlight the search string when displaying the resulting record. Localizable applications commonly store the length of the string to select in the `matchCustom` field for this purpose. Some multi-byte character encodings represent certain characters both as a single-byte character and a multi-byte character. When the search is performed, the single-byte character is accurately matched against its multi-byte equivalent. For this reason, the length of the string searched for does not always equal the length of the matching string. Applications that support being localized to multi-byte character sets often set the `matchCustom` field to the length of the matching string in the call to `FindSaveMatch()` so that they know the length of the string to select.

sysAppLaunchCmdGoToURL

Applications can respond to this launch code to retrieve and display the specified URL.

The parameter block for this launch command is simply a pointer to a string containing the URL.

This launch code may be sent in the following instances:

- If the Wireless Internet Feature Set is, applications can send this launch code directly to the Web Clipping Application Viewer application.
- If 4.0 New Feature Set is present, the `ExgRequest()` function launches an application with this launch code if it

cannot find an exchange library that is registered for the URL it has received. To receive the launch code, the application must first use `ExgRegisterDatatype()` to register for a URL scheme.

IMPORTANT: Implemented only if Wireless Internet Feature Set is present.

---

sysAppLaunchCmdHandleSyncCallApp

This launch command is sent by the Desktop Link server when `SyncCallRemoteModule` is called from a conduit to request that the handheld application do some processing on the conduit's behalf.

Along with this launch code you receive a `sysAppLaunchCmdHandleSyncCallApp` parameter block which contains all of the information passed to `SyncCallRemoteModule` on the desktop plus the fields needed to pass the result back to the desktop. Pass the results back to the conduit by calling `DlkControl()`. See the comments section for `DlkControl` in the *Palm OS Programmer's API Reference* for an example of how to handle this launch code.

sysAppLaunchCmdHandleSyncCallApp Parameter Block

Prototype

```
typedef struct
SysAppLaunchCmdHandleSyncCallAppType {
   UInt16 pbSize;
   UInt16 action;
   void *paramP;
   UInt32 dwParamSize;
   void *dlRefP;
   Boolean handled;
   UInt8 reserved1;
   Err replyErr;
   UInt32 dwReserved1;
   UInt32 dwReserved2;
} SysAppLaunchCmdHandleSyncCallAppType;
```

Fields

`pbSize`
Size, in bytes, of this parameter block. Set to `sizeof(SysAppLaunchCmdHandleSyncCallAppType)`.

`action`
Call action ID (application-specific).

`paramP`
Pointer to parameter block (call action ID specific).

`dwParamSize`
Parameter block size, in bytes.

`dlRefP`
DesktopLink reference pointer. Supply this value in the `DlkCallAppReplyParamType` structure when calling `DlkControl()` with the `dlkCtlSendCallAppReply` control code.

`handled`
Initialized to `false` by DLServer; if handled, your application must set it to `true` (and your handler the handler must call `DlkControl` with the `dlkCtlSendCallAppReply`control code). If your handler is not going to send a reply back to the conduit, leave this field set to `false`, in which case the DesktopLink Server will send the default "unknown request" reply.

`reserved1`
Reserved. Set to `NULL`.

`replyErr`
Error code returned from the call to `DlkControl()` with the `dlkCtlSendCallAppReply` control code.

`dwReserved1`
Reserved. Set to `NULL`.

`dwReserved2`
Reserved. Set to `NULL`.

sysAppLaunchCmdInitDatabase

This launch code is sent by the Desktop Link server in response to a request to create a database. It is sent to the application whose creator ID matches that of the requested database.

The most frequent occurrence of this is when a `'data'` database is being installed or restored from the desktop. In this case, HotSync creates a new database on the device and passes it to the application via a `sysAppLaunchCmdInitDatabase` command, so that the application can perform any required initialization. HotSync will then transfer the records from the desktop database to the device database.

When a Palm OS application crashes while a database is installed using HotSync, the reason may be that the application is not handling the `sysAppLaunchCmdInitDatabase` command properly. Be especially careful not to access global variables.

The system will create a database and pass it to the application for initialization. The application must perform any initialization required, then pass the database back to the system, unclosed.

sysAppLaunchCmdInitDatabase Parameter Block

Prototype

```
typedef struct {
   DmOpenRef dbP;
   UInt32 creator;
   UInt32 type;
   UInt16 version;
} SysAppLaunchCmdInitDatabaseType;
```

Fields

`dbP`
: Database reference.

`creator`
: Database creator.

`type`
: Database type.

`version`
: Database version.

sysAppLaunchCmdLookup

The system or an application sends this launch command to retrieve information from another application. In contrast to Find, there is a level of indirection; for example, this launch code could be used to retrieve the phone number based on input of a name.

This functionality is currently supported by the standard Palm OS Address Book.

Applications that decide to handle this launch code must search their database for the string the user entered and perform the match operation specified in the launch code's parameter block.

If an application wants to allow its users to perform lookup in other applications, it has to send it properly, including all information necessary to perform the match. An example for this is in `Address.c` and `AppLaunchCmd.h`, which are included in your SDK.

sysAppLaunchCmdLookup Parameter Block

The parameter block is defined by the application that supports this launch code. See `AppLaunchCmd.h` for an example.

IMPORTANT: Implemented only if 2.0 New Feature Set is present.

---

sysAppLaunchCmdNormalLaunch

The system or an application sends this launch code to launch an application. Upon receiving this launch code the application should perform a full launch and display its user interface. See "Responding to Normal Launch" on page 23 for details on launching your application in response to this launch code.

Global variables are available upon receipt of this launch code.

See Also `sysAppLaunchCmdCardLaunch`

sysAppLaunchCmdNotify

The system or an application sends this launch code to notify applications that an event has occurred. The parameter block specifies the type of event that occurred, as well as other pertinent information. To learn which notifications are broadcast by the system, see the chapter titled "Notifications" in this book.

IMPORTANT: Implemented only if Notification Feature Set is present.

sysAppLaunchCmdNotify Parameter Block

The `SysNotifyParamType` structure declared in `NotifyMgr.h` defines the format of this launch code's parameter block. See its description in the "Notifications" chapter.

sysAppLaunchCmdOpenDB

You can send this launch code to the Web Clipping Application Viewer application to launch the application and cause it to open and display a Palm™ query application stored on the device. This is the same mechanism that the Launcher uses to launch query applications.

IMPORTANT: Implemented only if Wireless Internet Feature Set is present.

sysAppLaunchCmdOpenDB Parameter Block

Prototype

```
typedef struct {
   UInt16 cardNo;
   LocalID dbID;
} SysAppLaunchCmdOpenDBType;
```

Fields

`cardNo`
: Card number of database to open.

`dbID`
: Database id of database to open.

sysAppLaunchCmdPanelCalledFromApp

`sysAppLaunchCmdPanelCalledFromApp` and `sysAppLaunchCmdReturnFromPanel` allow an application to let users change preferences without switching to the Preferences application. For example, for the calculator, you may launch the Formats preferences panel, set up a number format preference, then directly return to the calculator that then uses the new format.

`sysAppLaunchCmdPanelCalledFromApp` lets a preferences panel know whether it was switched to from the Preferences application or whether an application invoked it to make a change. The panel may be a preference panel owned by the application or a system preferences panel.

Examples of these system panels that may handle this launch code are:

- Network panel (called from network applications)
- Modem panel (called if modem selection is necessary)

All preferences panels must handle this launch code. If a panel is launched with this command, it should:

- Display a Done button.
- *Not* display the panel-switching pop-up trigger used for navigation within the preferences application.

IMPORTANT: Implemented only if 2.0 New Feature Set is present.

---

sysAppLaunchCmdReturnFromPanel

This launch code is used in conjunction with `sysAppLaunchCmdPanelCalledFromApp`. It informs an application that the user is done with a called preferences panel. The system passes this launch code to the application when a previously-called preferences panel exists.

IMPORTANT: Implemented only if 2.0 New Feature Set is present.

sysAppLaunchCmdSaveData

Instructs the application to save all current data. For example, before the system performs a global find, an application should save all data.

Any application that supports the Find command and that can have buffered data should support this launch code. The system sends this launch code to the currently active application before it begins the search. The application receiving this launch code should respond by saving all buffered data so that the search is able to find matches in the text just entered.

sysAppLaunchCmdSaveData Parameter Block

Prototype

```
typedef struct {
   Boolean uiComing;
   UInt8 reserved1;
} SysAppLaunchCmdSaveDataType;
```

Fields

`uiComing`
: `true` if the system dialog is displayed before launch code arrives.

`reserved1`
: Reserved for future use.

sysAppLaunchCmdSyncNotify

This launch code is sent to applications to inform them that a HotSync operation has occurred.

This launch code is sent only to applications whose databases were changed during the HotSync operation. (Installing the application database itself is considered a change.) The record database(s) must have the same creator ID as the application in order for the system to know which application to send the launch code to.

This launch code provides a good opportunity to update, initialize, or validate the application's new data, such as resorting records, setting alarms, and so on.

Because applications only receive `sysAppLaunchCmdSyncNotify` when their databases are updated, this launch code is not a good place to perform any operation that must occur after every HotSync operation. Instead, you may register to receive the `sysNotifySyncFinishEvent` on systems that have the Notification Feature Set. This notification is sent at the end of a HotSync operation, and it is sent to all applications registered to receive it, whether the application's data changed or not. Note that there is also a `sysNotifySyncStartEvent` notification.

sysAppLaunchCmdSystemLock

Launch code sent to the system-internal security application to lock the device.

As a rule, applications don't need to do respond to this launch code. If an application replaces the system-internal security application, it must handle this launch code.

IMPORTANT: Implemented only if 2.0 New Feature Set is present.

sysAppLaunchCmdSystemReset

Launch code to respond to system soft or hard reset.

Applications can respond to this launch code by performing initialization, indexing, or other setup that they need to do when the system is reset. For more information about resetting the device, see "System Boot and Reset" in the *Palm OS Programmer's Companion*, vol. I.

sysAppLaunchCmdSystemReset Parameter Block

Prototype

```
typedef struct {
   Boolean hardReset;
   Boolean createDefaultDB;
} SysAppLaunchCmdSystemResetType;
```

Fields

`hardReset`
: `true` if system was `hardReset`. `false` if system was `softReset`.

`createDefaultDB`
: If `true`, application has to create default database.

sysAppLaunchCmdTimeChange

Launch code to respond to a time change initiated by the user.

Applications that are dependent on the current time or date need to respond to this launch code. For example, an application that sets alarms may want to cancel an alarm or set a different one if the system time changes.

On systems that have the Notification Feature Set, applications should register to receive the `sysNotifyTimeChangeEvent` notification instead of responding to this launch code. The `sysAppLaunchCmdTimeChange` launch code is sent to all applications. The `sysNotifyTimeChangeEvent` notification is sent only to applications that have specifically registered to receive it, making it more efficient than `sysAppLaunchCmdTimeChange`.

sysAppLaunchCmdURLParams

This launch code is sent from the Web Clipping Application Viewer application to launch another application.

The parameter block consists of a pointer to a special URL string, which the application must know how to parse. The string is the URL used to launch the application and may contain encoded parameters.

An application launched with this code may or may not have access to global variables, static local variables, and code segments other

than segment 0 (in multi-segment applications). It depends on the URL that caused the Web Clipping Application Viewer to send this launch code. If this launch code results from a `palm` URL, then globals are available. If the launch code results from a `palmcall` URL, then globals are not available.

The best way to test if you have global variable access is to test the `sysAppLaunchFlagNewGlobals` launch flag sent with this launch code. If this is flag is set, then you have global variable access.

IMPORTANT: Implemented only if Wireless Internet Feature Set is present.

Launch Flags

When an application is launched with any launch command, it also is passed a set of launch flags.

An application may decide to ignore the flags even if it handles the launch code itself. For applications that decide to use the launch flags, the following table provides additional information:

Table 1.3 Launch Flags

Flag	Functionality
`sysAppLaunchFlagNewGlobals`	Set when the system has created and initialized a new globals world for the application. Implies new owner ID for memory chunks.
`sysAppLaunchFlagUIApp`	Set when a UI application is being launched.
`sysAppLaunchFlagSubCall`	Set when the application is calling its entry point as a subroutine call. This tells the launch code that it's OK to keep the A5 (globals) pointer valid through the call. If this flag is set, it indicates that the application is already running as the current application.

IMPORTANT: Applications should never set launch flags when sending a launch code to another application. They should only be set by the system. In particular, note that you should never pass `sysAppLaunchFlagNewGlobals` as a launch flag for `SysAppLaunch`. If you do and you make repeated calls to `SysAppLaunch`, the system eventually runs out of owner IDs, and the new application fails to launch.

2

Palm OS Events

Palm OS® events are structures (defined in the header files `Event.h`, `SysEvent.h`, and `INetMgr.h`) that the system passes to the application when the user interacts with the graphical user interface. Chapter 3, "Event Loop," on page 53, in the *Palm OS Programmer's Companion*, vol. I discusses in detail how this works. This chapter provides reference-style information about each event. First it shows the types used by Palm OS events. Then it discusses the following events in alphabetical order:

Event	UI Object
`appStopEvent`	N.A.
`ctlEnterEvent`, `ctlExitEvent`, `ctlRepeatEvent`, `ctlSelectEvent`	Control
`daySelectEvent`	N.A.
`fldChangedEvent`, `fldEnterEvent`, `fldHeightChangedEvent`	Field
`frmCloseEvent`, `frmGotoEvent`, `frmLoadEvent`, `frmOpenEvent`, `frmSaveEvent`, `frmUpdateEvent`, `frmTitleEnterEvent`, `frmTitleSelectEvent`	Form
`frmObjectFocusLostEvent`, `frmObjectFocusTakeEvent`	Form object.
`frmGadgetEnterEvent`, `frmGadgetMiscEvent`	Extended gadget
`inetSockReadyEvent`, `inetSockStatusChangeEvent`	N.A. (INetLib)
`keyDownEvent`, `keyUpEvent`	N.A.
`lstEnterEvent`, `lstExitEvent`, `lstSelectEvent`	List
`menuEvent`, `menuOpenEvent`, `menuCloseEvent`, `menuCmdBarOpenEvent`	Menu
`nilEvent`	N.A.

Event	UI Object
penDownEvent, penMoveEvent, penUpEvent	N.A. (pen)
popSelectEvent	Popup (Control)
sclEnterEvent, sclRepeatEvent, sclExitEvent	Scroll bar
tblEnterEvent, tblExitEvent, tblSelectEvent	Table
winDisplayChangedEvent, winEnterEvent, winExitEvent	Window

Event Data Structures

eventsEnum

Purpose The `eventsEnum` enum specifies the possible event types.

Prototype

```
typedef enum {
  nilEvent = 0,
  penDownEvent,
  penUpEvent,
  penMoveEvent,
  keyDownEvent,
  winEnterEvent,
  winExitEvent,
  ctlEnterEvent,
  ctlExitEvent,
  ctlSelectEvent,
  ctlRepeatEvent,
  lstEnterEvent,
  lstSelectEvent,
  lstExitEvent,
  popSelectEvent,
  fldEnterEvent,
  fldHeightChangedEvent,
  fldChangedEvent,
  tblEnterEvent,
  tblSelectEvent,
  daySelectEvent,
  menuEvent,
```

```
  appStopEvent = 22,
  frmLoadEvent,
  frmOpenEvent,
  frmGotoEvent,
  frmUpdateEvent,
  frmSaveEvent,
  frmCloseEvent,
  frmTitleEnterEvent,
  frmTitleSelectEvent,
  tblExitEvent,
  sclEnterEvent,
  sclExitEvent,
  sclRepeatEvent,
  tsmFepModeEvent,

  attnIndicatorEnterEvent,
  attnIndicatorSelectEvent,

  menuCmdBarOpenEvent = 0x0800,
  menuOpenEvent,
  menuCloseEvent,
  frmGadgetEnterEvent,
  frmGadgetMiscEvent,

  firstINetLibEvent = 0x1000,
  firstWebLibEvent = 0x1100,

  keyUpEvent = 0x4000,
  keyHoldEvent = 0x4001,
  frmObjectFocusTakeEvent = 0x4002,
  frmObjectFocusLostEvent = 0x4003,

  firstUserEvent = 0x6000,
  lastUserEvent  = 0x7FFF
} eventsEnum;
```

Each of these event types is discussed in alphabetical order below.

EventType Struct

Purpose The `EventType` structure contains all the data associated with a system event. All event types have some common data. Most events

also have data specific to those events. The specific data uses a union that is part of the `EventType` data structure. The union can have up to 16 bytes of specific data.

The common data is documented below the structure. The Event Reference section gives details on the important data associated with each type of event.

Prototype

```
typedef struct {
  eventsEnum eType;
  Boolean penDown;
  UInt8 tapCount;
  Int16 screenX;
  Int16 screenY;
  union{
    ...
  } data;
} EventType;
```

Fields

`eType`
: One of the `eventsEnum` constants. Specifies the type of the event.

`penDown`
: `true` if the pen was down at the time of the event, otherwise `false`.

`tapCount`
: The number of taps received at this location. This value is used mainly by fields. When the user taps in a text field, two taps selects a word, and three taps selects the entire line.

`screenX`
: Window-relative position of the pen in pixels (number of pixels from the left bound of the window).

`screenY`
: Window-relative position of the pen in pixels (number of pixels from the top left of the window).

`data`
: The specific data for an event, if any. The data is a union, and its exact contents depend on the `eType` field. The Event Reference section in this chapter shows what the `data` field contains for each event.

NOTE: Remember that the `data` field is part of the access path to an identifier in the `EventType` structure. As an example, the code to access the `controlID` field of a `ctlEnterEvent` would be:

```
EventType *event;
//...
if (event->data.ctlEnter.controlID ==
            MyAppLockButton)
```

Compatibility The `tapCount` field is only defined if 3.5 New Feature Set is present. Because of the `tapCount` field, it's particularly important that you clear the event structure before you use it to add a new event to the queue in Palm OS 3.5 and higher. Otherwise, the `tapCount` value may be incorrect for the new event.

EventPtr Typedef

Purpose The `EventPtr` defines a pointer to an `EventType`.

Prototype

```
typedef EventType *EventPtr;
```

Event Reference

appStopEvent

When the system wants to launch a different application than the one currently running, the event manager sends this event to request the current application to terminate. In response, an application has to exit its event loop, close any open files and forms, and exit.

If an application doesn't respond to this event by exiting, the system can't start the other application.

attnIndicatorEnterEvent

Purpose The function `FrmHandleEvent()` sends this event when it receives a `penDownEvent` within the bounds of the attention indicator.

For this event, the `data` field contains the following structure:

Prototype

```
struct attnIndicatorEnter {
  UInt16 formID;
} attnIndicatorEnter;
```

Fields

`formID`
: Developer-defined ID of the form.

Compatibility This event is only supported if 5.0 New Feature Set is present.

attnIndicatorSelectEvent

Purpose The control function `FrmHandleEvent()` sends this event. When `FrmHandleEvent()` receives a `attnIndicatorEnterEvent`, it tracks the pen until the pen is lifted. If the pen is lifted within the bounds of the attention indicator, an `attnIndicatorSelectEvent` is added to the event queue.

For this event, the `data` field contains the following structure:

Prototype

```
struct attnIndicatorSelect {
  UInt16 formID;
} attnIndicatorSelect;
```

Fields

`formID`
: Developer-defined ID of the form.

Compatibility This event is only supported if 5.0 New Feature Set is present.

ctlEnterEvent

Purpose The control routine `CtlHandleEvent()` sends this event when it receives a `penDownEvent` within the bounds of a control.

For this event, the `data` field contains the following structure:

Prototype

```
struct ctlEnter {
  UInt16 controlID;
  struct ControlType *pControl;
} ctlEnter;
```

Fields

`controlID`
: Developer-defined ID of the control.

`pControl`
: Pointer to a control structure (`ControlType`).

ctlExitEvent

Purpose

The control routine `CtlHandleEvent()` sends this event. When `CtlHandleEvent` receives a `ctlEnterEvent`, it tracks the pen until the pen is lifted from the display. If the pen is lifted within the bounds of a control, a `ctlSelectEvent` is added to the event queue; if not, a `ctlExitEvent` is added to the event queue. The `penDown`, `screenX`, and `screenY` fields of the `EventType` structure are set appropriately for the `ctlExitEvent`. As well, the `data` field contains the following structure:

Prototype

```
struct ctlExit {
  UInt16 controlID;
  struct ControlType *pControl;
} ctlExit;
```

Fields

`controlID`
: Developer-defined ID of the control.

`pControl`
: Pointer to a control structure (`ControlType`).

ctlRepeatEvent

Purpose

The control routine `CtlHandleEvent()` sends this event. When `CtlHandleEvent` receives a `ctlEnterEvent` in a repeating button (tREP) or a feedback slider control (tslf), it sends a `ctlRepeatEvent`. When `CtlHandleEvent` receives a `ctlRepeatEvent` in a repeating button, it sends another `ctlRepeatEvent` if the pen remains down within the bounds of the control for 1/2 second beyond the last `ctlRepeatEvent`.

When `CtlHandleEvent` receives a `ctlRepeatEvent` in a feedback slider control, it sends a `ctlRepeatEvent` each time the slider's thumb moves by at least one pixel. Feedback sliders do not send `ctlRepeatEvents` at regular intervals like repeating buttons do.

If you return `true` in response to a `ctlRepeatEvent`, it stops the `ctlRepeatEvent` loop. No further `ctlRepeatEvents` are sent.

For this event, the `data` field contains the following structure:

Prototype

```
struct ctlRepeat {
  UInt16 controlID;
  struct ControlType *pControl;
  UInt32 time;
  UInt16 value;
} ctlRepeat;
```

Fields

`controlID`
: Developer-defined ID of the control.

`pControl`
: Pointer to a control structure (`ControlType`).

`time`
: System-ticks count when the event is added to the queue.

`value`
: Current value if the control is a feedback slider.

Compatibility

The `value` field is only present if 3.5 New Feature Set is present.

ctlSelectEvent

Purpose

The control routine `CtlHandleEvent()` sends this event. When `CtlHandleEvent` receives a `ctlEnterEvent`, it tracks the pen until the pen is lifted. If the pen is lifted within the bounds of the same control it went down in, a `cltSelectEvent` is added to the event queue; if not, a `ctlExitEvent` is added to the event queue.

It usually doesn't matter whether you return `true` or `false` from your event handler since the operating system doesn't handle this event. The default event handler for popup triggers does handle this event, however, so you must return `false` in this instance to ensure that the list is actually displayed.

For this event, the `data` field contains the following structure:

Prototype

```
struct ctlSelect {
  UInt16  controlID;
  struct ControlType *pControl;
  Boolean on;
  UInt8   reserved1;
  UInt16  value;
} ctlSelect;
```

Fields

`controlID`
: Developer-defined ID of the control.

`pControl`
: Pointer to a control structure (`ControlType`).

`on`
: `true` when the control is depressed; otherwise, `false`.

`reserved1`
: Unused.

`value`
: Current value if the control is a slider.

Compatibility

The `value` field is only present if 3.5 New Feature Set is present.

daySelectEvent

Purpose

The system-internal `DayHandleEvent` routine, which handles events in the day selector object, handles this event. When the day selector object displays a calendar month, the user can select a day by tapping on it.

This event is sent when the pen touches and is lifted from a day number.

For this event, the `data` field contains the following structure:

Prototype

```
struct daySelect {
  struct DaySelectorType *pSelector;
  Int16 selection;
  Boolean useThisDate;
  UInt8 reserved1;
} daySelect;
```

Fields

`pSelector`
: Pointer to a day selector structure (`DaySelectorType`).

`selection`
: Not used.

`useThisDate`
: Set to `true` to automatically use the selected date.

`reserved1`
: Unused.

fldChangedEvent

Purpose

The field routine `FldHandleEvent()` sends this event when the text of a field has or might have been scrolled. This event actually can be triggered from any call to the field code that causes scrolling to happen; most often, this happens during `FldHandleEvent`.

When `FldHandleEvent` receives a `fldEnterEvent`, it positions the insertion point and tracks the pen until it's lifted. Text is selected (highlighted) appropriately as the pen is dragged.

For this event, the `data` field contains the following structure:

Prototype

```
struct fldChanged {
  UInt16 fieldID;
  struct FieldType *pField;
} fldChanged;
```

Parameters

`fieldID`
: Developer-defined ID of the field.

`pField`
: Pointer to a field structure (`FieldType`).

fldEnterEvent

Purpose

The field routine `FldHandleEvent()` sends this event when the field receives a `penDownEvent` within the bounds of a field. For this event, the `data` field contains the following structure:

Prototype

```
struct fldEnter {
  UInt16 fieldID;
  struct FieldType *pField;
} fldEnter;
```

Fields

`fieldID`
: Developer-defined ID of the field.

`pField`
: Pointer to a field structure (`FieldType`).

fldHeightChangedEvent

Purpose

Several field routines, including `FldHandleEvent()`, send this event when the number of lines in the field changes. These functions send a `fldHeightChangedEvent` to notify your application that the height of a field needs to change.

If the field is contained in a table, the table's code handles the `fldHeightChangedEvent`. If the field is directly on a form, your application code should handle the `fldHeightChangedEvent` itself. The form code does not handle the event for you.

For this event, the `data` field contains the following structure:

Prototype

```
struct fldHeightChanged {
  UInt16 fieldID;
  struct FieldType *pField;
  Int16 newHeight;
  UInt16 currentPos;
} fldHeightChanged;
```

Fields

`fieldID`
: Developer-defined ID of the field.

`pField`
: Pointer to a field structure (`FieldType`).

`newHeight`
: New visible height of the field, in number of lines.

`currentPos`
: Current position of the insertion point.

frmCloseEvent

Purpose The form routines `FrmGotoForm()` and `FrmCloseAllForms()` send this event. `FrmGotoForm` sends a `frmCloseEvent` to the currently active form; `FrmCloseAllForms` sends a `frmCloseEvent` to all forms an application has loaded into memory. If an application doesn't intercept this event, the routine `FrmHandleEvent()` erases the specified form and releases any memory allocated for it.

For this event, the `data` field contains the following structure:

Prototype

```
struct frmClose {
  UInt16 formID;
} frmClose;
```

Fields

`formID`
: Developer-defined ID of the form.

frmGadgetEnterEvent

Purpose The function `FrmHandleEvent()` sends this event when there is a `penDownEvent` within the bounds of an extended gadget. The gadget handler function (see `FormGadgetHandlerType()`) should handle this event.

For this event, the `data` field contains the following structure:

Prototype

```
struct gadgetEnter {
  UInt16 gadgetID;
  struct FormGadgetType *gadgetP;
} gadgetEnter;
```

Fields

`gadgetID`
: Developer-defined ID of the gadget.

`gadgetP`
: Pointer to the `FormGadgetType` object representing this gadget.

Compatibility Implemented only if 3.5 New Feature Set is present.

frmGadgetMiscEvent

Purpose An application may choose to send this event when it needs to pass information to an extended gadget. The `FrmHandleEvent()` function passes `frmGadgetMiscEvents` on to the extended gadget's handler function (see `FormGadgetHandlerType()`).

For this event, the `data` field contains the following structure:

Prototype

```
struct gadgetMisc {
  UInt16 gadgetID;
  struct FormGadgetType *gadgetP;
  UInt16 selector;
  void *dataP;
} gadgetMisc;
```

Fields

`gadgetID`
: Developer-defined ID of the gadget.

`gadgetP`
: Pointer to the `FormGadgetType` object representing this gadget.

`selector`
: Any necessary integer value to pass to the gadget handler function.

`dataP`
: A pointer to any necessary data to pass to the gadget handler function.

Compatibility Implemented only if 3.5 New Feature Set is present.

frmGotoEvent

Purpose An application may choose to send itself this event when it receives a `sysAppLaunchCmdGoTo` launch code. `sysAppLaunchCmdGoTo` is generated when the user selects a record in the global find facility. Like `frmOpenEvent`, `frmGotoEvent` is a request that the application initialize and draw a form, but this event provides extra information so that the application may display and highlight the matching string in the form.

The application is responsible for handling this event.

For this event, the `data` field contains the following structure:

Prototype

```
struct frmGoto {
  UInt16 formID;
  UInt16 recordNum;
  UInt16 matchPos;
  UInt16 matchLen;
  UInt16 matchFieldNum;
  UInt32 matchCustom;
} frmGoto;
```

Fields

`formID`
: Developer-defined ID of the form.

`recordNum`
: Index of record containing the match string.

`matchPos`
: Position of the match.

`matchLen`
: Length of the matched string.

`matchFieldNum`
: Number of the field the matched string was found in.

`matchCustom`
: Application-specific information. You might use this if you need to provide extra information to locate the matching string within the record.

frmLoadEvent

Purpose

The form routines `FrmGotoForm()` and `FrmPopupForm()` send this event. It's a request that the application load a form into memory.

The application is responsible for handling this event. In response to this event, applications typically initialize the form, make it active, and set the event handler.

For this event, the `data` field contains the following structure:

Prototype

```
struct frmLoad {
  UInt16 formID;
} frmLoad;
```

Fields `formID`
Developer-defined ID of the form.

frmObjectFocusLostEvent

Purpose In some devices, a 5-way navigation button generates this event as the navigation focus moves in a form. The event is sent after an object has lost the focus.

For this event, the `data` field contains the following structure:

Prototype

```
struct frmObjectFocusLost{
  UInt16 formID;
  UInt16 objectID;
  UInt32 dispatchHint;
} frmObjectFocusLost;
```

Fields `formID`
Developer-defined ID of the form.

`objectID`
ID of the object in the tab order.

`dispatchHint`
System use only.

Comments The system's internal focus structures have already been updated when this event is sent. Therefore, it simply initiates the redrawing of the object that lost the focus.

When `FrmHandleEvent()` receives this event, it will call the type-specific handler of the UI object associated with the event. Usually, the type-specific handler will draw the UI object associated with the event in its normal state and then return true.

Compatibility Some devices that have 5-way navigation button—such as the Handspring Treo 600 Smartphone—generate this event. Consult the licensee's developer documentation to see whether this event is generated by a particular device.

frmObjectFocusTakeEvent

Purpose

In some devices, a 5-way navigation button generates this event as the navigation focus moves in a form. The event is sent when an object should grab the focus.

For this event, the `data` field contains the following structure:

Prototype

```
struct frmObjectFocusTake{
  UInt16 formID;
  UInt16 objectID;
  UInt32 dispatchHint;
} frmObjectFocusTake;
```

Fields

`formID`
: Developer-defined ID of the form.

`objectID`
: ID of the object in the tab order.

`dispatchHint`
: System use only.

Comments

The system's internal focus structures are updated when this event is *handled*, not when it is sent.

When `FrmHandleEvent()` receives this event, it will call the type-specific handler of the UI object associated with the event. The type-specific handler must call `FrmSetFocus()` for the associated object, draw the object in its focused state, and then return true. `FrmSetFocus()` updates the system's internal focus structures and then sends a `frmObjectFocusLostEvent`.

Compatibility

Some devices that have a 5-way navigation button—such as the Handspring Treo 600 Smartphone—generate this event. Consult the licensee's developer documentation to see whether this event is generated by a particular device.

frmOpenEvent

Purpose

The form routines `FrmGotoForm()` and `FrmPopupForm()` send this event. It is a request that the application initialize and draw a form.

The application is responsible for handling this event.

For this event, the `data` field contains the following structure:

Prototype

```
struct frmOpen {
  UInt16 formID;
} frmOpen;
```

Parameters

`formID`
Developer-defined ID of the form.

frmSaveEvent

The form routine `FrmSaveAllForms()` sends this event. It is a request that the application save any data stored in a form.

The application is responsible for handling this event.

No data is passed with this event.

frmTitleEnterEvent

Purpose

The control routine `FrmHandleEvent()` sends this event when it receives a `penDownEvent` within the bounds of the title of the form. Note that only the written title, not the whole title bar is active.

For this event, the `data` field contains the following structure:

Prototype

```
struct frmTitleEnter {
  UInt16 formID;
  } frmTitleEnter;
```

Fields

`formID`
Developer-defined ID of the form.

frmTitleSelectEvent

Purpose

The control routine `FrmHandleEvent()` sends this event. `FrmHandleEvent()` receives a `frmTitleEnterEvent`, it tracks the pen until the pen is lifted. If the pen is lifted within the bounds of the active same title bar region, a `frmTitleSelectEvent` is added to the event queue.

For this event, the `data` field contains the following structure:

Prototype

```
struct frmTitleSelect {
  UInt16 formID;
} frmTitleSelect;
```

Fields

`formID`
: Developer-defined ID of the form.

Compatibility

In Palm OS version 3.5 and higher, `FrmHandleEvent()` responds to `frmTitleSelectEvent`. Its response is to enqueue a `keyDownEvent` with a `vchrMenu` character to display the form's menu.

frmUpdateEvent

Purpose

The form routine `FrmUpdateForm()`, or in some cases the routine `FrmEraseForm()`, sends this event when it needs to redraw the region obscured by the form being erased.

Generally, the region obscured by a form is saved and restored by the form routines without application intervention. However, in cases where the system is running low on memory, the form's routine may not save obscured regions itself. In that case, the application adds a `frmUpdateEvent` to the event queue. The form receives the event and redraws the region using the `updateCode` value.

An application can define its own `updateCode` and then use this event to also trigger behavior in another form, usually when changes made to one form need to be reflected in another form.

For this event, the `data` field contains the following structure:

Prototype

```
struct frmUpdate {
  UInt16 formID;
  UInt16 updateCode;
} frmUpdate;
```

Fields

`formID`
: Developer-defined ID of the form.

`updateCode`
: The reason for the update request. `FrmEraseForm` sets this code to `frmRedrawUpdateCode`, which indicates that the entire form needs to be redrawn. Application developers can

define their own `updateCode`. The `updateCode` is passed as a parameter to `FrmUpdateForm()`.

inetSockReadyEvent

Purpose This event is returned only by `INetLibGetEvent()` (not `EvtGetEvent`) when the Internet library determines that a socket has data ready for an `INetLibSockRead()`.

For this event, the `data` field contains the following structure:

Prototype

```
struct  {
  MemHandle sockH;
  UInt32 context;
  Boolean inputReady;
  Boolean outputReady;
} inetSockReady;
```

Fields

`sockH`
: Socket handle of the socket that this event refers to.

`context`
: Not used.

`inputReady`
: `true` when the socket has data ready for the `INetLibSockRead()` call.

`outputReady`
: Not used.

The `penDown`, `tapCount`, `screenX` and `screenY` fields are *not* valid for Internet library events and should be ignored.

Compatibility Implemented only if Wireless Internet Feature Set is present.

inetSockStatusChangeEvent

Purpose This event is returned only by `INetLibGetEvent()` (not `EvtGetEvent`) when the Internet library determines that a socket has data ready for an `INetLibSockRead()`.

For this event, the `data` field contains the following structure:

Prototype

```
struct  {
  MemHandle sockH;
  UInt32 context;
  UInt16 status;
  Err sockErr;
}inetSockStatusChange;
```

Fields

`sockH`
: Socket handle of the socket that this event refers to.

`context`
: Not used.

`status`
: Current status of the socket. This is one of the `INetStatusEnum` constants.

`sockErr`
: Reason for failure of the last operation, if any. The current socket error can be cleared by calling `INetLibSockStatus()`.

The `penDown`, `tapCount`, `screenX` and `screenY` fields are *not* valid for Internet library events and should be ignored.

Compatibility

Implemented only if Wireless Internet Feature Set is present.

keyDownEvent

Purpose

This event is sent by the system when the user enters a Graffiti® or Graffiti 2 character, presses one of the buttons below the display, or taps one of the icons in the icon area; for example, the Find icon.

For this event, the `data` field contains the structure defined below. Access the information stored in the `data` field in this way:

```
wchar32_t = eventP->data.keyDown.chr;
```

Prototype

```
struct _KeyDownEventType {
  WChar chr;
  UInt16 keyCode;
  UInt16 modifiers;
};
```

Fields

`chr`
: The character code.

`keyCode`
: Unused.

`modifiers`
: 0, or one or more of the following values:

  `shiftKeyMask`
  : Graffiti or Graffiti 2 is in case-shift mode.

  `capsLockMask`
  : Graffiti is in cap-shift mode. (Cap-shift mode is not supported in Graffiti 2.)

  `numLockMask`
  : Graffiti or Graffiti 2 is in numeric-shift mode.

  `commandKeyMask`
  : The glyph was the menu command glyph or a virtual key code.

  `optionKeyMask`
  : Not implemented. Reserved.

  `controlKeyMask`
  : Not implemented. Reserved.

  `autoRepeatKeyMask`
  : Event was generated due to auto-repeat.

  `doubleTapKeyMask`
  : Not implemented. Reserved.

  `poweredOnKeyMask`
  : The key press caused the system to be powered on.

  `appEvtHookKeyMask`
  : System use only.

  `libEvtHookKeyMask`
  : System use only.

  `willSendKeyUpMask`
  : Sent only from hardware keys. Indicates that a `keyUpEvent` will be sent and a `keyHoldEvent` could be sent.

Compatibility

The `willSendKeyUpMask` applies only to certain devices that support a 5-way navigation button—such as the Handspring Treo

600 Smartphone. Consult the licensee's developer documentation to see whether this version of the `keyDownEvent` is generated by a particular device.

IMPORTANT: If the Graffiti 2 Feature Set is present, your application may receive a key down event for a character that is actually meant to be the first stroke of a multi-stroke character. See "Event Translation: Pen Strokes to Key Events" on page 64 of the *Palm OS Programmer's Companion*, vol. I for more information.

---

keyHoldEvent

Purpose

Sent when the user holds a hard key on the device.

For this event, the `data` field contains the structure defined below. Access the information stored in the `data` field in this way:

```
wchar32_t = eventP->data.keyHold.chr;
```

Prototype

```
struct _keyHoldEventType {
  WChar chr;
  UInt16 keycode;
  UInt16 modifiers;
};
```

Fields

`chr`
: The character code.

`keyCode`
: Unused.

`modifiers`
: 0, or one or more of the key modifier constants listed under `modifiers` in `keyDownEvent`.

Comments

This event is sent when a hardware key is held for one second. (Note that the one-second timing may be modified by a Palm OS licensee.)

Unlike `keyDownEvent`, `keyHoldEvent` is sent for characters corresponding to hardware buttons only. If the device contains a hardware keyboard, holding a key results in a `keyDownEvent` and then, one second later, a `keyHoldEvent`, and, ultimately, a

keyUpEvent. If the user is using a software keyboard, the application receives only the keyDownEvent.

Only one keyHoldEvent is sent for each press-and-hold of the key. You do not get a keyHoldEvent for each additional second that the key is held.

A keyHoldEvent is sent only for the most recently pressed key. For instance, if a key is pressed and held, and then another key is pressed within a second and before the first key is released, the following happens: a keyHoldEvent is *not* sent for the first key, but a keyHoldEvent *is* sent for the second key if it is held for a full second. Note that a keyUpEvent will be generated for both the first and second key as each key is released.

Compatibility Some devices that support a 5-way navigation button—such as the Handspring Treo 600 Smartphone—generate this event. Consult the licensee's developer documentation to see whether this event is generated by a particular device.

See Also keyDownEvent, keyUpEvent

keyUpEvent

Purpose Sent when the user releases a hard key on the device.

For this event, the data field contains the structure defined below. Access the information stored in the data field in this way:

```
wchar32_t = eventP->data.keyUp.chr;
```

Prototype

```
struct _keyUpEventType {
  WChar chr;
  UInt16 keycode;
  UInt16 modifiers;
};
```

Fields

chr
: The character code.

keyCode
: Unused.

modifiers
: 0, or one or more of the key modifier constants listed under modifiers in keyDownEvent.

Comments Unlike keyDownEvent, the keyUpEvent is sent for characters corresponding to hardware buttons only. If the device contains a hardware keyboard, the application receives keyDownEvent and keyUpEvent for each character typed. If the user is using a software keyboard, the application only receives the keyDownEvent.

Compatibility Some devices that support a 5-way navigation button—such as the Handspring Treo 600 Smartphone—generate this event. Consult the licensee's developer documentation to see whether this event is generated by a particular device.

lstEnterEvent

Purpose The list routine LstHandleEvent() sends this event when it receives a penDownEvent within the bounds of a list object.

For this event, the data field contains the following structure:

Prototype

```
struct lstEnter {
  UInt16 listID;
  struct ListType *pList;
  Int16 selection;
} lstEnter;
```

Fields

listID
: Developer-defined ID of the list.

pList
: Pointer to a list structure (ListType).

selection
: Unused.

lstExitEvent

Purpose The list routine LstHandleEvent() sends this event. When LstHandleEvent receives a lstEnterEvent, it tracks the pen until the pen is lifted. If the pen is lifted within the bounds of a list, a lstSelectEvent is added to the event queue; if not, a lstExitEvent is added to the event queue.

For this event, the `data` field contains the following structure:

Prototype

```
struct lstExit {
  UInt16 listID;
  struct ListType *pList;
} lstExit;
```

Fields

`listID`
: Developer-defined ID of the list.

`pList`
: Pointer to a list structure (`ListType`).

lstSelectEvent

Purpose

The list routine `LstHandleEvent()` sends this event. When `LstHandleEvent` receives a `lstEnterEvent`, it tracks the pen until the pen is lifted. If the pen is lifted within the bounds of a list, a `lstSelectEvent` is added to the event queue; if not, a `lstExitEvent` is added to the event queue.

Note that popup lists don't generate a `lstSelectEvent`. Instead, they generate a `popSelectEvent`.

For this event, the `data` field contains the following structure:

Prototype

```
struct lstSelect {
  UInt16 listID;
  struct ListType *pList;
  Int16 selection;
} lstSelect;
```

Fields

`listID`
: Developer-defined ID of the list.

`pList`
: Pointer to a list structure (`ListType`).

`selection`
: Item number (zero-based) of the new selection.

menuCloseEvent

This event is not currently used.

menuCmdBarOpenEvent

Purpose The menu routine `MenuHandleEvent()` sends this event when the user enters the menu shortcut keystroke, causing the command toolbar to be displayed at the bottom of the screen. Applications might respond to this event by calling `MenuCmdBarAddButton()` to add custom buttons to the command toolbar. Shared libraries or other non-application code resources can add buttons to the toolbar by registering to receive the `sysNotifyMenuCmdBarOpenEvent` notification.

For this event, the `data` field contains the following structure:

Prototype

```
struct menuCmdBarOpen {
  Boolean preventFieldButtons;
  UInt8 reserved;
} menuCmdBarOpen;
```

Fields

`preventFieldButtons`
: If `true`, the field manager does not add the standard cut, copy, paste, and undo buttons when the focus is on a field. If `false`, the field adds the buttons.

`reserved`
: Unused.

To prevent the command toolbar from being displayed, respond to this event and return `true`. Returning `true` prevents the form manager from displaying the toolbar.

Compatibility Implemented only if 3.5 New Feature Set is present.

menuEvent

Purpose The menu routine `MenuHandleEvent()` sends this event:

- When the user selects an item from a pull-down menu
- When the user selects a menu command using the command stroke followed by an available command; for example, Command-C for copy
- When the user taps one of the buttons on the command toolbar and the button is set up to generate a `menuEvent`.

For this event, the `data` field contains the following structure:

Prototype

```
struct menu {
  UInt16 itemID;
} menu;
```

Fields

`itemID`
Item ID of the selected menu command.

menuOpenEvent

Purpose

The menu routine `MenuHandleEvent()` sends this event when a new active menu has been initialized. A menu becomes active the first time the user taps the Menu silk-screen button or taps the form's titlebar, and it might need to be re-initialized and reactivated several times during the life of an application.

A menu remains active until one of the following happens:

- A `FrmSetMenu()` call changes the active menu on the form.
- A new form, even a modal form or alert panel, becomes active.

Suppose a user selects your application's About item from the Options menu then clicks the OK button to return to the main form. When the About dialog is displayed, it becomes the active form, which causes the main form's menu state to be erased. This menu state is not restored when the main form becomes active again. The next time the user requests the menu, it must be initialized again, so `menuOpenEvent` is sent again.

Applications might respond to this event by adding, hiding, or un-hiding menu items using the functions `MenuAddItem()`, `MenuHideItem()`, or `MenuShowItem()`.

A `menuCloseEvent` is defined by the system, but it is not currently sent. If you need to perform some cleanup (such as closing a resource) after the menu item you added is no longer needed, do so in response to `frmCloseEvent`.

For this event, the `data` field contains the following structure:

Prototype

```
struct menuOpen {
  UInt16 menuRscID;
  Int16 cause;
} menuOpen;
```

Fields

`menuRscID`
: Resource ID of the menu.

`cause`
: Reason for opening the menu. If `menuButtonCause`, the user tapped the Menu silkscreen button or tapped the form's titlebar, and the menu is going to be displayed. If `menuCommandCause`, the user entered the command keystroke, so the menu is becoming active without being displayed.

Compatibility

Implemented only if 3.5 New Feature Set is present.

nilEvent

A `nilEvent` is useful for animation, polling, and similar situations.

The event manager sends this event when there are no events in the event queue. This can happen if the routine `EvtGetEvent()` is passed a time-out value (a value other than `evtWaitForever`, -1). If `EvtGetEvent` is unable to return an event in the specified time, it returns a `nilEvent`. Different Palm OS versions and different devices can send `nilEvents` under different circumstances, so you might receive a `nilEvent` even before the timeout has expired.

penDownEvent

The event manager sends this event when the pen first touches the digitizer.

The following data is passed with the event:

Fields

`penDown`
: Always `true`.

`tapCount`
: The number of taps received at this location.

`screenX`
: Window-relative position of the pen in pixels (number of pixels from the left bound of the window).

`screenY`
: Window-relative position of the pen in pixels (number of pixels from the top left of the window).

penMoveEvent

Purpose The event manager sends this event when the pen is moved on the digitizer. Note that several kinds of UI objects, such as controls and lists, track the movement directly, and no `penMoveEvent` is generated.

The following data is passed with the event:

Fields

`penDown`
: Always `true`.

`tapCount`
: The number of taps received at this location.

`screenX`
: Window-relative position of the pen in pixels (number of pixels from the left bound of the window).

`screenY`
: Window-relative position of the pen in pixels (number of pixels from the top left of the window).

penUpEvent

Purpose The event manager sends this event when the pen is lifted from the digitizer. Note that several kinds of UI objects, such as controls and lists, track the movement directly, and no `penUpEvent` is generated.

For this event, the `data` field contains the following structure:

Prototype

```
struct _PenUpEventType {
  PointType start;
  PointType end;
};
```

Fields

`start`
: Display-relative start point of the stroke.

`end`
: Display-relative end point of the stroke.

In addition, the following data is passed with this event:

`penDown`
: Always `false`.

`tapCount`
: The number of taps received at this location.

`screenX`
: Window-relative position of the pen in pixels (number of pixels from the left bound of the window).

`screenY`
: Window-relative position of the pen in pixels (number of pixels from the top left of the window).

popSelectEvent

Purpose

The form routine `FrmHandleEvent()` sends this event when the user selects an item in a popup list.

For this event, the `data` field contains the following structure:

Prototype

```
struct popSelect {
  UInt16 controlID;
  struct ControlType *controlP;
  UInt16 listID;
  struct ListType *listP;
  Int16 selection;
  Int16 priorSelection;
} popSelect;
```

Fields

`controlID`
: Developer-defined ID of the resource.

`controlP`
: Pointer to the control structure (`ControlType`) of the popup trigger object.

`listID`
: Developer-defined ID of the popup list object.

`listP`
: Pointer to the list structure (`ListType`) of the popup list object.

`selection`
: Item number (zero-based) of the new list selection.

`priorSelection`
: Item number (zero-based) of the prior list selection.

sclEnterEvent

Purpose The routine `SclHandleEvent()` sends this event when it receives a `penDownEvent` within the bounds of a scroll bar.

Applications usually don't have to handle this event.

For this event, the `data` field contains the following structure:

Prototype

```
struct sclEnter {
  UInt16 scrollBarID;
  struct ScrollBarType *pScrollBar;
} sclEnter;
```

Fields

`scrollBarID`
: Developer-defined ID of the scroll bar resource.

`pScrollBar`
: Pointer to the scroll bar structure.

sclExitEvent

Purpose The routine `SclHandleEvent()` sends this event when the user lifts the pen from the scroll bar.

Applications that want to implement non-dynamic scrolling should wait for this event, then scroll the text using the values provided in `value` and `newvalue`.

Note that this event is sent regardless of previous `sclRepeatEvents`. If, however, the application has implemented dynamic scrolling, it doesn't have to catch this event.

For this event, the `data` field contains the following structure:

Prototype

```
struct sclExit {
  UInt16 scrollBarID;
  struct ScrollBarType *pScrollBar;
  Int16 value;
  Int16 newValue;
} sclExit;
```

Fields

`scrollBarID`
: Developer-defined ID of the scroll bar resource.

`pScrollBar`
: Pointer to the scroll bar structure.

`value`
: Initial position of the scroll bar

`newvalue`
: New position of the scroll bar. Given `value` and `newValue`, you can actually tell how much you have scrolled.

sclRepeatEvent

Purpose

The routine `SclHandleEvent()` sends this event when the pen is continually held within the bounds of a scroll bar.

Applications that implement dynamic scrolling should watch for this event. In dynamic scrolling, the display is updated as the user drags the scroll bar (not after the user releases the scroll bar).

For this event, the `data` field contains the following structure:

Prototype

```
struct sclRepeat {
  UInt16 scrollBarID;
  struct ScrollBarType *pScrollBar;
  Int16 value;
  Int16 newValue;
  Int32 time;
} sclRepeat;
```

Fields

`scrollBarID`
: Developer-defined ID of the scroll bar resource.

`pScrollBar`
: Pointer to the scroll bar structure.

`value`
: Initial position of the scroll bar.

`newValue`
: New position of the scroll bar. Given `value` and `newValue`, you can actually tell how much you have scrolled.

`time`
: System-ticks count when the event is added to the queue to determine when the next event should occur.

tblEnterEvent

Purpose

The table routine `TblHandleEvent()` sends this event when it receives a `penDownEvent` within the bounds of an active item in a table object.

For this event, the `data` field contains the following structure:

Prototype

```
struct tblEnter {
  UInt16 tableID;
  struct TableType *pTable;
  Int16 row;
  Int16 column;
} tblEnter;
```

Fields

`tableID`
: Developer-defined ID of the table.

`pTable`
: Pointer to a table structure (`TableType`).

`row`
: Row of the item.

`column`
: Column of the item.

tblExitEvent

Purpose

The table routine `TblHandleEvent()` sends this event. When `TblHandleEvent` receives a `tblEnterEvent`, it tracks the pen until it's lifted from the display. If the pen is lifted within the bounds of the same item it went down in, a `tblSelectEvent` is added to

the event queue; if not, a `tblExitEvent` is added to the event queue.

For this event, the `data` field contains the following structure:

Prototype

```
struct tblExit {
  UInt16 tableID;
  struct TableType *pTable;
  Int16 row;
  Int16 column;
} tblExit;
```

Fields

`tableID`
: Developer-defined ID of the table.

`pTable`
: Pointer to a table structure (`TableType`).

`row`
: Row of the item.

`column`
: Column of the item.

tblSelectEvent

Purpose

The table routine `TblHandleEvent()` sends this event. When `TblHandleEvent` receives a `tblEnterEvent`, it tracks the pen until the pen is lifted from the display. If the pen is lifted within the bounds of the same item it went down in, a `tblSelectEvent` is added to the event queue; if not, a `tblExitEvent` is added to the event queue.

For this event, the `data` field contains the following structure:

Prototype

```
struct tblSelect {
  UInt16 tableID;
  struct TableType *pTable;
  Int16 row;
  Int16 column;
} tblSelect;
```

Fields

`tableID`
: Developer-defined ID of the table.

`pTable`
: Pointer to a table structure (`TableType`).

`row`
: Row of the item.

`column`
: Column of the item.

winDisplayChangedEvent

Purpose

Posted by `PINSetInputAreaState` after the dynamic input area has been opened or closed. Normally, the user opens and closes the dynamic input area, but applications may also do so.

Applications may respond to the event by redrawing the active form in the available space.

By the time the application receives this event, the OS has already changed the bounds of the display window as appropriate to the state of the input area. The application must resize its active form's window and relayout the form accordingly.

No data is passed with this event.

Compatibility

Sent only if the Pen Input Manager Feature Set is present and returns `pinAPIVersion1_1`.

winEnterEvent

Purpose

The event manager sends this event when a window becomes the active window. This can happen in two ways: a call to `WinSetActiveWindow()` is issued (`FrmSetActiveForm()` calls this routine), or the user taps within the bounds of a window that is visible but not active. All forms are windows, but not all windows are forms; for example, the menu bar is a window but not a form.

For this event, the `data` field contains the following structure:

Prototype

```
struct _WinEnterEventType {
  WinHandle enterWindow;
  WinHandle exitWindow;
};
```

Fields

`enterWindow`
: Handle to the window we are entering. If the window is a form, then this is a pointer to a `FormType` structure; if not, it's a pointer to a `WindowType` structure.

`exitWindow`
: Handle to the window we are exiting, if there is currently an active window, or zero if there is no active window. If the window is a form, then this is a pointer to a `FormType` structure; if not, it's a pointer to a `WindowType` structure.

winExitEvent

Purpose

This event is sent by the event manager when a window is deactivated. A window is deactivated when another window becomes the active window (see `winEnterEvent`).

For this event, the `data` field contains the following structure:

Prototype

```
struct _WinExitEventType {
  WinHandle enterWindow;
  WinHandle exitWindow;
};
```

Fields

`enterWindow`
: Handle to the window we are entering. If the window is a form, then this is a pointer to a `FormType` structure; if not, it's a pointer to a `WindowType` structure.

`exitWindow`
: Handle to the window we are exiting. If the window is a form, then this is a pointer to a `FormType` structure; if not, it's a pointer to a `WindowType` structure.

3

Notifications

This chapter provides detailed information about the notifications declared in the header file `NotifyMgr.h`. Notifications are broadcast to inform applications, shared libraries, system extensions, or other code resources of certain system-level or application-level events.

Notifications are similar to application launch codes, but they differ from launch codes in the following ways:

- The system broadcasts notifications only to interested parties. To register to receive a notification, use `SysNotifyRegister()`.
- Notifications can be sent to non-applications.

See the "Notification Manager" chapter in this book and the section "Notifications" on page 30 of the *Palm OS Programmer's Companion*, vol. I for more information on receiving and handling notifications.

Table 3.1 Notification Constants

Constant	Description
`cncNotifyProfileEvent`	The connection profile used by the Connection Panel has changed.
`kTelTelephonyNotification`	A telephony event has occurred.
Socket Notification	A network socket change has occurred.
`sysExternalConnectorAttachEvent`	A device has been attached to an external connector.
`sysExternalConnectorDetachEvent`	A device has been detached from an external connector.
`sysNotifyAntennaRaisedEvent`	The antenna has been raised on a Palm VII™ series device.

Table 3.1 Notification Constants *(continued)*

Constant	Description
sysNotifyAppLaunchingEvent	An application is about to be launched.
sysNotifyAppQuittingEvent	An application has just quit.
sysNotifyCardInsertedEvent	An expansion card has been inserted into the expansion slot.
sysNotifyCardRemovedEvent	An expansion card has been removed from the expansion slot.
sysNotifyDBCreatedEvent	A database has been created.
sysNotifyDBChangedEvent	Database info has been set on a database, such as with DmSetDatabaseInfo().
sysNotifyDBDeletedEvent	A database has been deleted.
sysNotifyDBDirtyEvent	A database has been opened for write or in some other way has been made modifiable.
sysNotifyDeleteProtectedEvent	The Launcher has attempted to delete a protected database.
sysNotifyDeviceUnlocked	The user has unlocked the device.
sysNotifyDisplayChangeEvent	The color table or bit depth has changed.
sysNotifyDisplayResizedEvent	The dynamic input area has opened or closed.
sysNotifyEarlyWakeupEvent	The system is starting to wake up.
sysNotifyEventDequeuedEvent	An event has been removed from the event queue with EvtGetEvent.
sysNotifyForgotPasswordEvent	The user has tapped the Lost Password button in the Security application.

Table 3.1 Notification Constants *(continued)*

Constant	Description
sysNotifyGotUsersAttention	The Attention Manager has informed the user of an event.
sysNotifyGsiDrawIndicator	The system is about to draw the shift indicator.
sysNotifyHelperEvent	An application has requested that a particular service be performed.
sysNotifyIdleTimeEvent	The system is idle and is about to doze.
sysNotifyInputAreaDrawingEvent	The system is about to draw the dynamic input area.
sysNotifyInputAreaPendownEvent	The system is about to post a penDownEvent for the dynamic input area.
sysNotifyInsPtEnableEvent	The insertion point is being enabled or disabled.
sysNotifyIrDASniffEvent	Not used.
sysNotifyKeyboardDialogEvent	The keyboard dialog is about to be displayed.
sysNotifyLateWakeupEvent	The system has finished waking up.
sysNotifyLocaleChangedEvent	The system locale has changed.
sysNotifyMenuCmdBarOpenEvent	The system is about to display the menu command toolbar.
sysNotifyNetLibIFMediaEvent	The system has been connected to or disconnected from the network.
sysNotifyPhoneEvent	Reserved for future use.
sysNotifyPOSEMountEvent	System use only.
sysNotifyProcessPenStrokeEvent	The user has made a pen stroke on the silkscreen portion of the digitizer.

Table 3.1 Notification Constants *(continued)*

Constant	Description
sysNotifyResetFinishedEvent	The system has finished a reset.
sysNotifyRetryEnqueueKey	The Attention Manager has failed to post a virtual character to the key queue.
sysNotifySelectDay	The system needs to request that the user pick a particular date in the calendar.
sysNotifySleepNotifyEvent	The system is about to go to sleep.
sysNotifySleepRequestEvent	The system has decided to go to sleep.
sysNotifySyncFinishEvent	A HotSync® operation has just completed.
sysNotifySyncStartEvent	A HotSync operation is about to begin.
sysNotifyTimeChangeEvent	The system time has just changed.
sysNotifyVirtualCharHandlingEvent	A virtual character is being handled.
sysNotifyVolumeMountedEvent	A file system has been mounted.
sysNotifyVolumeUnmountedEvent	A file system has been unmounted.

Notification Data Structures

SysNotifyParamType Typedef

Purpose The `SysNotifyParamType` structure contains all of the data associated with a notification. This structure is passed as the parameter block for the `sysAppLaunchCmdNotify` launch code or as a parameter to the notification callback function. All notifications have some common data. Most notifications also have data specific to that notification. The specific data is pointed to by the `notifyDetailsP` field.

The common data for each notification is documented below the following structure declaration. The Notification Reference section gives details on the important data associated with each type of notification.

Prototype

```
typedef struct SysNotifyParamType {
  UInt32 notifyType;
  UInt32 broadcaster;
  void *notifyDetailsP;
  void *userDataP;
  Boolean handled;
  UInt8 reserved2;
} SysNotifyParamType;
```

Fields

`notifyType`
: The type of event that occurred. See Notification Reference.

`broadcaster`
: The creator ID of the application that broadcast the notification, or `sysNotifyBroadcasterCode` if the system broadcast the event.

`notifyDetailsP`
: Pointer to data specific to this notification.

`userDataP`
: Custom data that your notification handler requires. You create this data and pass it to `SysNotifyRegister()`.

`handled`
: Set to `true` if the notification has been handled; set to `false` otherwise. In some cases, `handled` is treated as a bit field that notification handlers can use to indicate that certain conditions are true.

`reserved2`
: Reserved for future use.

Notification Reference

cncNotifyProfileEvent

The `cncNotifyProfileEvent` is broadcast whenever a connection profile has been created, modified, or deleted and after a request has been made to update the connection profile list.

The `notifyDetailsP` field informs the notification handler of the type of change that was made. Register for the `cncNotifyProfileEvent` if your application maintains its own list of connection profiles that it should keep current or if it should help the Connection Panel maintain its list.

cncNotifyProfileEvent Specific Data

`notifyDetailsP` points to a `CncProfileNotifyDetailsType` structure.

Prototype

```
typedef struct _CncProfileNotifyDetailsTag {
  UInt16 version;
  UInt32 profileID;
  UInt16 deviceKind;
  UInt16 request;
} CncProfileNotifyDetailsType;
```

Fields

`version`
: The current version of this structure. Use the `kCncProfileNotifyCurrentVersion` constant to find out what the current version is.

`profileID`
: The ID of the modified connection profile.

`deviceKind`
: Device kind of the profile. This can be one of the following constants:

  `kCncDeviceKindSerial`
  : Serial connection profile.

  `kCncDeviceKindModem`
  : Modem profile.

`kCncDeviceKindPhone`
Phone profile.

`kCncDeviceKindLocalNetwork`
LAN profile.

`request`
The action that was performed. This can be one of the following constants:

`kCncNotifyCreateRequest`
The profile has been created.

`kCncNotifyDeleteRequest`
The profile is about to be deleted.

`kCncNotifyModifyRequest`
The profile has been modified.

`kCncNotifyUpdateListRequest`
A HotSync operation or system reset has just occurred. The notification handler should update the Connection Panel's list.

If a profile has been created or modified, the `request` field also contains a flag indicating how the current profile is to be set:

`kCncBecomeCurrentModifier`
The new profile should be made the current profile.

`kCncNotifyAlertUserModifier`
The user is prompted to set the current profile.

Compatibility Implemented only if 4.0 New Feature Set is present.

kTelTelephonyNotification

The `kTelTelephonyNotification` is broadcast when the Palm OS receives an event from the phone, such as an incoming SMS message or an incoming phone call.

kTelTelephonyNotification Specific Data

`notifyDetailsP` points to a `TelNotificationType` structure.

Prototype

```
typedef struct _TelNotificationType {
   UInt32 notificationData;
   UInt32 notificationData2;
   UInt32 timeStamp;
   UInt16 notificationId;
   UInt8 priority;
} TelNotificationType
```

Fields

`notificationData`
: Data specific to the `notificationId`.

`notificationData2`
: Data specific to a `notificationId`.

`timeStamp`
: Time the event occurred, expressed as the number of seconds elapsed since 12:00 A.M. on January 1, 1904.

`notificationId`
: One of the constants described in "Notification Identifiers" on page 1680 describing the type of event that occurred.

`priority`
: One of the constants described in "Notification Priorities" on page 1677.

Socket Notification

A socket notification is broadcast when the state of a network socket changes—for example, a socket is closed or a socket receives TCP data. This notification differs from the others described in this chapter in a significant way: there is no single constant value that identifies socket notification. Instead, each application must have a unique value for purposes of receiving socket notifications. Your application must pass this unique constant as the `notifyType` parameter to `SysNotifyRegister()`.

Socket notifications are used by Net Library to deliver socket notices. There is more to them than just the normal process of registering for a notification. For more details, see "Socket Notices" on page 188 of *Palm OS Programmer's Companion*, vol. II, *Communications*.

The `notifyDetailsP` field informs the notification handler what socket underwent a change and the nature of that change.

Socket Notification Specific Data

`notifyDetailsP` points to a `SysNotifyNetSocketType` structure.

Prototype

```
typedef struct SysNotifyNetSocketType {
  NetSocketRef socketRef;
  UInt32 condition;
} SysNotifyNetSocketType;
```

Fields

`socketRef`
: Integer representing the socket whose state change triggered the notification.

`condition`
: A bitfield describing the socket condition(s) that triggered the notification. The Socket Notice Trigger Conditions are constants defined for the bits of this field.

Compatibility Implemented only if 5.4 New Feature Set is present.

sysExternalConnectorAttachEvent

The `sysExternalConnectorAttachEvent` is broadcast when a USB cradle, RS-232 cradle or peripheral, a power cable, or a modem is attached to the universal connector. This notification is broadcast only on devices that have the universal connector.

sysExternalConnectorAttachEvent Specific Data

The `notifyDetailsP` field points to a `UInt16` that identifies which type of device was attached.

Compatibility Implemented only if 4.0 New Feature Set is present.

sysExternalConnectorDetachEvent

The `sysExternalConnectorDetachEvent` is broadcast when a USB cradle, a RS-232 cradle or peripheral, a power cable, or a

modem is detached from the universal connector. This notification is only broadcast on devices that have the universal connector.

sysExternalConnectorDetachEvent Specific Data

The `notifyDetailsP` field points to a `UInt16` that identifies which type of device was detached.

Compatibility Implemented only if 4.0 New Feature Set is present.

sysNotifyAntennaRaisedEvent

The `sysNotifyAntennaRaisedEvent` is broadcast by `SysHandleEvent()` when the antenna is raised on a Palm VII series device.

Register for this notification if you want to handle the antenna key down event. To ensure that no other code handles the antenna key down event after yours, set the `handled` parameter of the `SysNotifyParamType` structure to `true`.

sysNotifyAntennaRaisedEvent Specific Data

None.

Compatibility Implemented only if Notification Feature Set is present.

sysNotifyAppLaunchingEvent

The `sysNotifyAppLaunchingEvent` is broadcast before an application is launched with `sysAppLaunchCmdNormalLaunch`.

sysNotifyAppLaunchingEvent Specific Data

notifyDetailsP points to a `SysNotifyAppLaunchOrQuitType` structure.

Prototype

```
typedef struct SysNotifyAppLaunchOrQuitTag {
  UInt32 version;
  UInt32 dbID;
  UInt16 cardNo;
} SysNotifyAppLaunchOrQuitType;
```

Fields

`version`
The current version of this structure. The current version is 0.

`dbID`
The local ID of the application.

`cardNo`
The number of the card on which the application resides.

Compatibility

This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don't broadcast this notification. Palm OS Garnet releases do broadcast it. Later versions may or may not broadcast this notification.

sysNotifyAppQuittingEvent

The `sysNotifyAppQuittingEvent` is broadcast right after an application that was launched with `sysAppLaunchCmdNormalLaunch` quits.

sysNotifyAppLaunchingEvent Specific Data

`notifyDetailsP` points to a `SysNotifyAppLaunchOrQuitType` structure. See the description of `sysNotifyAppLaunchingEvent` for a description of this structure.

Compatibility

This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don't broadcast this notification. Palm OS Garnet releases do broadcast it. Later versions may or may not broadcast this notification.

sysNotifyCardInsertedEvent

The `sysNotifyCardInsertedEvent` is broadcast when an Expansion Manager card is inserted into a slot. When a new card is inserted, the Expansion Manager attempts to mount the volume on that card and plays a sound (indicating success or failure) once the attempt is complete.

Most applications will want to register for `sysNotifyVolumeMountedEvent` instead of this notification.

Register for `sysNotifyCardInsertedEvent` if you need to know when a card is inserted or if you want to prevent the Expansion Manager from performing its default handling of the notification.

To prevent the Expansion Manager from mounting the volume, set the `expHandledVolume` bit in the `handled` field. To prevent the Expansion Manager from playing the sound, set the `expHandledSound` bit in the `handled` field. For example:

```
cmdPBP->handled |= expHandledSound;
```

sysNotifyCardInsertedEvent Specific Data

`notifyDetailsP` points to a `UInt16` containing the slot reference number.

Compatibility Implemented only if 4.0 New Feature Set is present.

sysNotifyCardRemovedEvent

The `sysNotifyCardRemovedEvent` is broadcast when an Expansion Manager card is removed from a slot. When a card is removed, the Expansion Manager responds to this notification by playing a goodbye sound and then attempting to unmount the volume.

Most applications will want to register for `sysNotifyVolumeUnmountedEvent` instead of this notification. Register for `sysNotifyCardRemovedEvent` if you need to know when a card is removed or if you want to prevent the Expansion Manager from performing its default handling of the notification.

To prevent the Expansion Manager from unmounting the volume, set the `expHandledVolume` bit in the `handled` field. To prevent the Expansion Manager from playing the sound, set the `expHandledSound` bit in the `handled` field. For example:

```
cmdPBP->handled |= expHandledSound;
```

sysNotifyCardRemovedEvent Specific Data

`notifyDetailsP` points to a `UInt16` containing the slot reference number.

Compatibility Implemented only if 4.0 New Feature Set is present.

sysNotifyDBCreatedEvent

The `sysNotifyDBCreatedEvent` is broadcast sometime *after* a database is created with `DmCreateDatabase()`.

Register for this notification if you keep an internal list of databases that needs to be updated when a new database is created.

IMPORTANT: The `sysNotifyDB...Event` notifications are deferred notifications. So, for instance, if your application creates a database, opens it for write, and then renames it, all before `EvtGetEvent` is called, the three corresponding notifications will all go out together. A `sysNotifyDBDirtyEvent` handler would fail if it tried to open the database, since the database will already have been renamed. You must be aware of the ramifications of a deferred notification when writing your notification handler.

---

sysNotifyDBCreatedEvent Specific Data

`notifyDetailsP` points to a `SysNotifyDBCreatedType` structure.

Prototype

```
typedef struct SysNotifyDBCreatedTag {
   Char dbName[dmDBNameLength];
   UInt32 creator;
   UInt32 type;
   LocalID newDBID;
   UInt16 cardNo;
   Boolean resDB;
   UInt8 padding;
} SysNotifyDBCreatedType;
```

Fields

`dbName`
: Database name.

`creator`
: Database creator ID.

`type`
: Database type.

`newDBID`
: Local ID of the newly-created database.

`cardno`
: Card number upon which the database resides.

`resDB`
: `true` if the database is a resource database, `false` otherwise.

`padding`
: Structure padding byte.

Compatibility Implemented only if 5.0 New Feature Set is present.

sysNotifyDBChangedEvent

The `sysNotifyDBChangedEvent` is broadcast sometime *after* database info is set with `DmSetDatabaseInfo()`.

Register for this notification if you keep an internal list of databases that needs to be updated when database info changes.

IMPORTANT: The `sysNotifyDB...Event` notifications are deferred notifications. So, for instance, if your application creates a database, opens it for write, and then renames it, all before `EvtGetEvent` is called, the three corresponding notifications will all go out together. A `sysNotifyDBDirtyEvent` handler would fail if it tried to open the database, since the database will already have been renamed. You must be aware of the ramifications of a deferred notification when writing your notification handler.

---

sysNotifyDBChangeEvent Specific Data

`notifyDetailsP` points to a `SysNotifyDBChangedType` structure. The contents of `fields` in this structure indicates what about the database changed, and thus which of the other structure fields contain valid data.

Prototype

```
typedef struct SysNotifyDBChangedTag {
   Char dbName[dmDBNameLength];
   LocalID dbID;
   UInt32 creator;
   UInt32 type;
   UInt32 crDate;
   UInt32 modDate;
   UInt32 bckUpDate;
   UInt32 modNum;
   LocalID appInfoID;
   LocalID sortInfoID;
   UInt16 attributes;
   UInt16 cardNo;
   UInt16 version;
   UInt16 fields;
   Char oldName[dmDBNameLength];
   UInt32 oldCreator;
   UInt32 oldType;
   UInt16 oldAttributes;
   UInt16 padding;
} SysNotifyDBChangedType;
```

Fields

`dbName`
: New name of database.

`dbID`
: Database ID.

`creator`
: New database creator ID.

`type`
: New database type.

`crDate`
: New database creation date.

`modDate`
: New database modification date.

`bckUpDate`
: New database backup date.

`modNum`
: New database modification number.

`appInfoID`
New database application info block.

`sortInfoID`
New database sort info block.

`attributes`
New database attributes.

`cardNo`
Card number upon which the dabatase resides.

`version`
New database version.

`fields`
Flags that indicate what about the database changed, and thus which of the above fields are set. The constants that define the `fields` bits are:

```
DBChangedFieldSetName = 0x1
DBChangedFieldSetCreator = 0x2
DBChangedFieldSetType = 0x4
DBChangedFieldSetCrDate = 0x8
DBChangedFieldSetModDate = 0x10
DBChangedFieldSetBckUpDate = 0x20
DBChangedFieldSetModNum = 0x40
DBChangedFieldSetAppInfo = 0x80
DBChangedFieldSetSortInfo = 0x100
DBChangedFieldSetAttributes = 0x200
DBChangedFieldSetVersion = 0x400
```

`oldName`
Name of database prior to the call to `DmSetDatabaseInfo`.

`oldCreator`
Database creator ID prior to the call to `DmSetDatabaseInfo`.

`oldType`
Database type prior to the call to `DmSetDatabaseInfo`.

`oldAttributes`
: Database attributes prior to the call to `DmSetDatabaseInfo`.

`padding`
: Structure padding bytes.

Compatibility Implemented only if 5.0 New Feature Set is present.

sysNotifyDBDeletedEvent

The `sysNotifyDBDeletedEvent` is broadcast sometime *after* a database is removed from the device.

Register for this notification if you keep an internal list of databases that needs to be updated upon removal of a database. For example, the Attention Manager and Connection Manager register for this notification to maintain their internal lists of databases.

IMPORTANT: The `sysNotifyDB...Event` notifications are deferred notifications. So, for instance, if your application creates a database, opens it for write, and then renames it, all before `EvtGetEvent` is called, the three corresponding notifications will all go out together. A `sysNotifyDBDirtyEvent` handler would fail if it tried to open the database, since the database will already have been renamed. You must be aware of the ramifications of a deferred notification when writing your notification handler.

---

sysNotifyDBDeletedEvent Specific Data

`notifyDetailsP` points to a `SysNotifyDBDeletedType` structure.

Prototype

```
typedef struct SysNotifyDBDeletedTag {
  LocalID oldDBID;
  UInt16 cardNo;
  UInt16 attributes;
  Char dbName[dmDBNameLength];
  UInt32 creator;
  UInt32 type;
} SysNotifyDBDeletedType;
```

Fields

`oldDBID`
The local ID of the deleted database. This ID is no longer valid.

WARNING! The ID in `oldDBID` is invalid by the time the notification is broadcast. If you try to pass it to a Data Manager function, the system will crash.

`cardNo`
The number of the card on which the database resided.

`attributes`
The deleted database's attributes.

`dbName`
The name of the deleted database.

`creator`
The creator ID of the deleted database.

`type`
The type of the deleted database.

Compatibility Implemented only if 4.0 New Feature Set is present.

sysNotifyDBDirtyEvent

The `sysNotifyDBDirtyEvent` is broadcast sometime *after* a database is opened for write or in some other way has been made modifiable. Note that the database may not have actually been modified yet.

Register for this notification if you keep an internal list of databases that needs to be updated when a database becomes "dirty." For instance, upon reset the Launcher normally checks over such databases and updates its internal list.

IMPORTANT: The `sysNotifyDB...Event` notifications are deferred notifications. So, for instance, if your application creates a database, opens it for write, and then renames it, all before `EvtGetEvent` is called, the three corresponding notifications will all go out together. A `sysNotifyDBDirtyEvent` handler would fail if it tried to open the database, since the database will already have been renamed. You must be aware of the ramifications of a deferred notification when writing your notification handler.

sysNotifyDBDirtyEvent Specific Data

`notifyDetailsP` points to a `SysNotifyDBDirtyType` structure.

Prototype

```
typedef struct SysNotifyDBDirtyTag {
   Char dbName[dmDBNameLength];
   UInt32 creator;
   UInt32 type;
} SysNotifyDBDirtyType;
```

Fields

`dbName`
: Database name.

`creator`
: Database creator ID.

`type`
: Database type.

Compatibility

Implemented only if 5.0 New Feature Set is present.

sysNotifyDeleteProtectedEvent

The `sysNotifyDeleteProtectedEvent` is broadcast when the Launcher attempts to delete a database that has the protected flag set. The Launcher broadcasts the notification and then attempts to delete the database again. Any third party application that deletes databases should broadcast this notification as well.

Register for this notification if you have a protected database but you still want to allow users to delete your application or other code resource if they choose. A notification handler should check the information in the `notifyDetailsP` struct to see if its database is

the one being deleted. If so, it should respond to this notification to perform any necessary cleanup and to clear the protected flag. In this way, when the Launcher attempts to delete the database again, it will succeed. Note that if an application has multiple protected databases, this notification may be sent out more than once.

sysNotifyDeleteProtectedEvent Specific Data

`notifyDetailsP` points to a `SysNotifyDBInfoType` structure.

Prototype

```
typedef struct SysNotifyDBInfoTag {
  LocalID dbID;
  UInt16 cardNo;
  UInt16 attributes;
  Char dbName[dmDBNameLength];
  UInt32 creator;
  UInt32 type;
} SysNotifyDBInfoType;
```

Fields

`dbID`
: The local ID of the database to be deleted.

`cardNo`
: The number of the card on which the database resides.

`attributes`
: The database's attributes.

`dbName`
: The name of the database to be deleted.

`creator`
: The creator ID of the database to be deleted.

`type`
: The type of the database to be deleted.

Compatibility

Implemented only if 4.0 New Feature Set is present.

sysNotifyDeviceUnlocked

The `sysNotifyDeviceUnlocked` notification is broadcast by the Security application when the user unlocks the device. The notification is broadcast immediately after the device has finished unlocking.

If you display UI in response to the `sysNotifyLateWakeupEvent` notification, you should also register to receive the `sysNotifyDeviceUnlocked` notification. When a locked device receives the `sysNotifyLateWakeupEvent`, your UI should not be displayed if the device is waiting for the user to enter the password. The `sysNotifyDeviceUnlocked` notification is broadcast after the password is entered, which indicates that the user interface is ready.

sysNotifyDeviceUnlocked Specific Data

None.

Compatibility Implemented only if 4.0 New Feature Set is present.

sysNotifyDisplayChangeEvent

The `sysNotifyDisplayChangeEvent` is broadcast whenever the display mode changes. That is, either the color table has been set to use a specific palette using the `WinPalette()` function or the bit depth has changed using the `WinScreenMode()` function.

The `notifyDetailsP` field indicates how the bit depth changed. If the two values in the struct are equal, it means that the color palette has changed instead of the bit depth.

sysNotifyDisplayChangeEvent Specific Data

`notifyDetailsP` points to a `SysNotifyDisplayChangeDetailsType` structure.

Prototype

```
typedef struct {
  UInt32 oldDepth;
  UInt32 newDepth;
} SysNotifyDisplayChangeDetailsType;
```

Fields

`oldDepth`
: The old bit depth.

`newDepth`
: The new bit depth.

Compatibility Implemented only if Notification Feature Set is present.

sysNotifyDisplayResizedEvent

The `sysNotifyDisplayResizedEvent` notification is broadcast by `PINSetInputAreaState` after the dynamic input area or control bar has been opened or closed. Normally, the user opens and closes the dynamic input area and control bar, but applications may also do so, though this is not encouraged.

Applications may respond to the notification by redrawing the active form in the available space.

sysNotifyDisplayResizedEvent Specific Data

`notifyDetailsP` points to a `SysNotifyDisplayResizedDetailsType` structure.

Prototype

```
typedef struct SysNotifyDisplayResizedDetailsTag{
   RectangleType newBounds;
} SysNotifyDisplayResizedDetailsType;
```

Fields

`newBounds`
: The new bounds of the application area after the input area or control bar has been opened or closed. The application should draw the current form within these bounds.

Compatibility

Implemented only if the Pen Input Manager Feature Set is present.

sysNotifyEarlyWakeupEvent

The `sysNotifyEarlyWakeupEvent` is broadcast during `SysHandleEvent()` immediately after the system has finished sleeping. The screen may still be turned off, and the system may not fully wake up. It may simply handle an alarm or a battery charger event and go back to sleep. Most applications that need notification of a wakeup event will probably want to register for `sysNotifyLateWakeupEvent` instead.

IMPORTANT: This notification is *not* guaranteed to be broadcast. Thus, it is not suitable for applications where external hardware must be turned on when the system is powered on.

sysNotifyEarlyWakeupEvent Specific Data

None.

Compatibility Implemented only if Notification Feature Set is present.

sysNotifyEventDequeuedEvent

The `sysNotifyEventDequeuedEvent` is broadcast for each event removed from the event queue with `EvtGetEvent`.

WARNING! Be very careful about registering for this notification; it can result in significantly degraded system performance.

sysNotifyEventDequeuedEvent Specific Data

`notifyDetailsP` points to the dequeued event's `EventType` structure.

IMPORTANT: For speed, the event structure that `notifyDetailsP` points to uses system-native endianness. This means that you might need to byte-swap the structure's contents, depending on the endianness of the underlying operating system.

Compatibility This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don't broadcast this notification. Palm OS Garnet release do broadcast it. Later versions may or may not broadcast this notification.

sysNotifyForgotPasswordEvent

The `sysNotifyForgotPasswordEvent` is broadcast after the user taps the Lost Password button in the Security application. The notification is sent after the user has confirmed that all private records should be deleted but before the deletion actually occurs.

sysNotifyForgotPasswordEvent Specific Data

None.

Compatibility Implemented only if Notification Feature Set is present.

sysNotifyGotUsersAttention

The `sysNotifyGotUsersAttention` notification is broadcast when the Attention Manager has finished displaying or sounding its attention indicators (blinking, playing sounds, vibrating, and so on).

System extensions or shared libraries should register for this notification if they want to perform some extra effect or if they simply want to be informed of when the user's attention was received.

sysNotifyGotUsersAttention Specific Data

`notifyDetailsP` points to an `AttnNotifyDetailsType` structure.

Prototype

```
typedef struct {
  AttnFlagsType flags;
} AttnNotifyDetailsType;
```

Fields

`flags`
: The attention indicators that were used to get the user's attention. See `AttnFlagsType`.

Compatibility Implemented only if 4.0 New Feature Set is present.

sysNotifyGsiDrawIndicator

The `sysNotifyGsiDrawIndicator` notification is broadcast just before the system is about to draw the shift indicator. If you handle this notification, then the system assumes that the shift indicator has been drawn, so it skips its own drawing code.

If you need to find out where to draw the shift indicator, you can loop through the form objects until you find one of type `frmGraffitiStateObj`, and then get its position. Draw the shift indicator with a width of `kMaxGsiWidth` (9) and `kMaxGsiHeight` (10), as defined in `GraffitiShift.h`. Note that you must clear

and draw every pixel in this 9 by 10 rectangle if you handle the notification.

sysNotifyGsiDrawIndicator Specific Data

None.

Compatibility Implemented only if 5.3SC New Feature Set is present.

NOTE: Future Palm OS versions might not support this notification.

---

sysNotifyHelperEvent

The `sysNotifyHelperEvent` is broadcast by applications to request a service from another application. For example, the Address Book application broadcasts this notification to request that the Dial application dial a phone number. For the `sysNotifyHelperEvent`, the notification client (that is, the application or shared library that registers for the notification) is called a **helper**.

The application that broadcasts this notification specifies one of the action codes listed in Table 39.1 in Chapter 39, "Helper API." These action codes request all helper applications to enumerate (list the services they perform), validate (ensure that the service will succeed), and execute (perform the action). The helper responds to the notification by returning the required data in the appropriate portion of the `notifyDetailsP` structure and by setting the `handled` field to `true` or `false` to indicate the success or failure of the action.

For more information on this notification, see the section "Helper Notifications" on page 38 in the *Palm OS Programmer's Companion*, vol. I.

sysNotifyHelperEvent Specific Data

`notifyDetailsP` points to a `HelperNotifyEventType` structure.

Compatibility Implemented only if 4.0 New Feature Set is present.

sysNotifyIdleTimeEvent

The `sysNotifyIdleTimeEvent` is broadcast when the system is idle and is about to doze.

Compatibility This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don't broadcast this notification. Palm OS Garnet releases do broadcast it. Later versions may or may not broadcast this notification.

sysNotifyInputAreaDrawingEvent

The `sysNotifyInputAreaDrawingEvent` notification is broadcast just before the system is about to redraw the dynamic input area (on a device supporting a dynamic input area). If you handle this notification, then the system assumes that the requested portion of the input area has been redrawn and skips its own drawing code.

sysNotifyInputAreaDrawingEvent Specific Data

`notifyDetailsP` points to a `SysNotifyInputAreaDrawingDetailsType` structure, which is in little-endian format.

Prototype

```
typedef struct
SysNotifyInputAreaDrawingDetailsTag {
  RectangleType updateBounds;
  const BitmapType *bitmapP;
  Boolean selected;
  Boolean fullRedraw;
  UInt16 padding;
} SysNotifyInputAreaDrawingDetailsType;
```

Fields

`updateBounds`
: If the `fullRedraw` field is `false`, then this is the area to redraw, in native coordinates, relative to the input area window.

`bitmapP`
: A pointer to the bitmap that is about to be drawn by the system.

`selected`
: `true` if the area being redrawn is a button that the user is in the process of tapping and should therefore be drawn in its selected state.

`fullRedraw`
: `true` if the entire input area should be redrawn, or `false` if only a portion is to be redrawn. If `true`, then the `updateBounds` field can be ignored.

`padding`
: Padding; unused.

Compatibility Implemented only if 5.3SC New Feature Set is present.

NOTE: Future Palm OS versions might not support this notification.

sysNotifyInputAreaPendownEvent

The `sysNotifyInputAreaPendownEvent` notification is broadcast just before the system is about to handle a `penDownEvent` for the dynamic input area (on a device supporting a dynamic input area). If you handle this notification, then the system assumes that the event has been handled and skips its own event handling code.

sysNotifyInputAreaPendownEvent Specific Data

`notifyDetailsP` points to a `SysNotifyInputAreaPendownDetailType` structure, which is in little-endian format.

Prototype

```
typedef struct SysNotifyInputAreaPendownDetailTag
{
  PointType where;
} SysNotifyInputAreaPendownDetailType;
```

Fields

`where`
: The point where the pen first went down, in native coordinates, relative to the input area window.

Compatibility Implemented only if 5.3SC New Feature Set is present.

NOTE: Future Palm OS versions might not support this notification.

sysNotifyInsPtEnableEvent

The `sysNotifyInsPtEnableEvent` is broadcast at the start of `InsPtEnable`.

sysNotifyInsPtEnableEvent Specific Data

`notifyDetailsP` points to a Boolean: the `enableIt` parameter passed to `InsPtEnable`.

Compatibility This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don't broadcast this notification. Palm OS Garnet releases do broadcast it. Later versions may or may not broadcast this notification.

sysNotifyKeyboardDialogEvent

The `sysNotifyKeyboardDialogEvent` is broadcast whenever the system keyboard is displayed. It is intended to enable the replacement of `SysKeyboardDialog` function's user interface.

sysNotifyKeyboardDialogEvent Specific Data

`notifyDetailsP` points to the `KeyboardType` enum that indicates the mode in which the keyboard should be opened: alphabetic, numeric, or international.

Compatibility This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don't broadcast this notification. Palm OS Garnet releases do broadcast it. Later versions may or may not broadcast this notification.

sysNotifyLateWakeupEvent

The `sysNotifyLateWakeupEvent` is broadcast during `SysHandleEvent()` immediately after the device has finished waking up. This notification is sent at the late stage of wakeup, after the screen has been turned on. When this notification is broadcast, the system is guaranteed to fully wake up. Register for this notification if you need to perform startup tasks each time the system wakes up.

IMPORTANT: This notification is *not* guaranteed to be broadcast. Thus, it is unsuitable for applications where external hardware must be powered on when the device wakes up.

When the device receives this notification, it may be locked and waiting for the user to enter the password. If this is the case, you must wait for the user to unlock the device before you display a user interface. Therefore, if you intend to display a user interface when the device wakes up, you should make sure the device is not locked. If the device is locked, you should register for `sysNotifyDeviceUnlocked` notification and display your user interface when it is received. For example:

```
case sysNotifyLateWakeupEvent:
  if ((Boolean)
      PrefGetPreference(prefDeviceLocked)) {
    SysNotifyRegister(myCardNo, myDbID,
      sysNotifyDeviceUnlocked, NULL,
      sysNotifyNormalPriority, NULL);
  } else {
    HandleDeviceWakeup();
  }
case sysNotifyDeviceUnlocked:
  HandleDeviceWakeup();
```

Note that the `sysNotifyDeviceUnlocked` notification is only broadcast on Palm OS 4.0 and higher.

sysNotifyLateWakeupEvent Specific Data

None.

Compatibility Implemented only if Notification Feature Set is present.

sysNotifyLocaleChangedEvent

The `sysNotifyLocaleChangedEvent` is broadcast immediately after the system locale has changed. Currently, the user has the opportunity to change the locale only when the device first starts up and after a hard reset.

RAM-based applications and other code resources should obtain locale information by passing the `prefLocale` constant to `PrefGetPreference()`. They should not register for this notification. This notification is used by the built-in applications, which respond to it by rebuilding their default databases to use the newly selected language and character set.

sysNotifyLocaleChangedEvent Specific Data

`notifyDetailsP` points to a `SysNotifyLocaleChangedType` structure.

Prototype

```
typedef struct SysNotifyLocaleChangedTag {
  LmLocaleType oldLocale;
  LmLocaleType newLocale;
} SysNotifyLocaleChangedType;
```

Fields

`oldLocale`
: The old locale. See `LmLocaleType`.

`newLocale`
: The new locale.

Compatibility

Implemented only if 4.0 New Feature Set is present.

sysNotifyMenuCmdBarOpenEvent

The `sysNotifyMenuCmdBarOpenEvent` is broadcast during `MenuHandleEvent()` when it is about to display the menu shortcut command bar.

Register for this notification if you are writing a system extension (such as a "hack" installed with the HackMaster program) that needs to add a button to the menu command bar or to suppress the menu command bar. To add a button, call `MenuCmdBarAddButton()`. To suppress the command toolbar, set the `handled` field to `true`.

Applications that need to add their own buttons to the menu command bar should do so in response to a `menuCmdBarOpenEvent`. They should *not* register for this notification because an application should only add buttons if it is already the active application. The notification is sent after the event has been received, immediately before the command toolbar is displayed.

sysNotifyMenuCmdBarOpenEvent

None.

Compatibility Implemented only if Notification Feature Set is present.

sysNotifyNetLibIFMediaEvent

The `sysNotifyNetLibIFMediaEvent` is broadcast at the top of the event loop whenever the network interface makes the network connection active or inactive. The Network Panel uses this notification to decide whether the Connect button should be active.

Register for this notification if you need to know when the network connection is currently active.

sysNotifyNetLibIFMediaEvent Specific Data

`notifyDetailsP` contains a `SysNotifyNetLibIFMediaType` structure.

Prototype

```
typedef struct SysNotifyNetLibIFMediaTag {
  NetLibIFMediaEventNotificationTypeEnum eType;
  UInt32 ifCreator;
  UInt16 ifInstance;
} SysNotifyNetLibIFMediaType;
```

Fields

`eType`

One of the following values:

`netIFMediaUp`

The network connection is active. This is usually sent after the network interface has displayed UI indicating that a connection attempt is in progress.

`netIFMediaDown`
: The network connection is inactive. This is usually sent after the network interface has brought the connection down because an inactivity timeout value was reached.

`ifCreator`
: Creator ID of the network interface

`ifInstance`
: Instance number of the network interface.

Compatibility Implemented only if 4.0 New Feature Set is present.

sysNotifyProcessPenStrokeEvent

The `sysNotifyProcessPenStrokeEvent` is broadcast to enable custom recognition of strokes made on the silkscreen portion of the digitizer.

sysNotifyProcessPenStrokeEvent Specific Data

`notifyDetailsP` points to a `SysNotifyPenStrokeType` structure.

Prototype

```
typedef struct SysNotifyPenStrokeTag {
  UInt32 version;
  PointType startPt;
  PointType endPt;
} SysNotifyPenStrokeType;
```

Fields

`version`
: The current version of this structure. The current version is 0.

`startPt`
: Start point of stroke.

`endPt`
: End point of stroke.

Compatibility This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don't broadcast this notification. Palm OS Garnet releases do broadcast it. Later versions may or may not broadcast this notification.

sysNotifyResetFinishedEvent

The `sysNotifyResetFinishedEvent` is broadcast immediately after the system has finished a reset.

Because the notification registry is cleared upon a reset, only internal system components use this notification. Applications that need to be informed of a system reset can respond to the `sysAppLaunchCmdSystemReset` launch code.

sysNotifyResetFinishedEvent Specific Data

None.

Compatibility Implemented only if Notification Feature Set is present.

sysNotifyRetryEnqueueKey

The `sysNotifyRetryEnqueueKey` notification is broadcast at the top of the event loop if the Attention Manager has attempted to post a virtual character to the key queue and failed because the queue is full. The notification signals that the Attention Manager is going to retry enqueuing the virtual character until it is successful.

Most applications do not need to register for this notification. It is used only by the Attention Manager to schedule retries of enqueuing the virtual character. When enqueueing a virtual character fails, the Attention Manager retries at the top of the event loop. It uses this notification to schedule retries so that they occur even if the user switches applications.

sysNotifyRetryEnqueueKey Specific Data

`notifyDetailsP` points to a `WChar` containing the virtual character to be enqueued.

Compatibility Implemented only if 4.0 New Feature Set is present.

sysNotifySelectDay

The `sysNotifySelectDay` notification is broadcast when the `SelectDay()` function (as defined in `SelDay.h`) is called. If you

handle this notification, then the system assumes that an appropriate user interface has been presented to the user and (if `dayChanged` is `true`) uses `daySelectorP` for the requested date. This allows you to develop a locale-specific date selection user interface (beyond that already supported by the `SelectDay()` function).

`notifyDetailsP` points to a `SysNotifySelectDayDetailsType` structure, which is in little-endian format.

Prototype

```
typedef struct SysNotifySelectDayDetailsTag {
  DaySelectorType *daySelectorP;
  const Char *titleP;
  Boolean dayChanged;
  UInt8 padding[3];
} SysNotifySelectDayDetailsType;
```

Fields

`daySelectorP`
: A pointer to a `DaySelectorType` structure that on input specifies the date to select when the user interface first appears, and on output specifies the date selected by the user.

`titleP`
: A pointer to a text string that may be displayed as the title of the user interface to inform the user what the date they select will be used for.

`dayChanged`
: `true` if the user selected a different date in the user interface, or `false` if they canceled the user interface or didn't change the initially selected date.

`padding`
: Padding; unused.

Compatibility

Implemented only if 5.3SC New Feature Set is present.

NOTE: Future Palm OS versions might not support this notification.

sysNotifySleepNotifyEvent

The `sysNotifySleepNotifyEvent` is broadcast during `SysHandleEvent()` immediately before the system is put to sleep. After the broadcast is complete, the system is put to sleep.

Register for this notification if you have a small amount of cleanup that needs to be performed before the system goes to sleep. It is recommended that you not perform any sort of prolonged activity, such as displaying an alert panel that requests confirmation, in response to a sleep notification. If you do, the alert might be displayed long enough to trigger another auto-off event, which could be detrimental to other handlers of this notification.

If your code is in the middle of a lengthy computation and needs to defer sleep, it should register for the `sysNotifySleepRequestEvent` instead.

IMPORTANT: This notification is *not* guaranteed to be broadcast. For example, if the system goes to sleep because the user removes the batteries, sleep notifications are not sent. Thus, these notifications are unsuitable for applications where external hardware must be shut off to conserve power before the system goes to sleep.

sysNotifySleepNotifyEvent Specific Data

None.

Compatibility Implemented only if Notification Feature Set is present.

sysNotifySleepRequestEvent

The `sysNotifySleepRequestEvent` is broadcast during `SysHandleEvent()` processing when the system has decided to go to sleep.

Register for this notification if you need to delay the system from going to sleep while your code performs a lengthy operation, such as disconnecting from the network. The system checks the `deferSleep` value when each notification handler returns. If it is nonzero, it cancels the sleep event.

After you defer sleep, your code is free to finish what it was doing. When it is finished, you must allow the system to continue with the sleep event. To do so, create a `keyDownEvent` with the `resumeSleepChr` and the command key bit set (to signal that the character is virtual) and add it to the event queue. When the system receives this event, it will again broadcast the `sysNotifySleepRequestEvent` to all clients. If `deferSleep` is 0 after all clients return, then the system knows it is safe to go to sleep, and it broadcasts the `sysNotifySleepNotifyEvent` to all of its clients.

Note that you may receive this notification several times before the system goes to sleep because notification handlers can delay the system sleep and resume it later.

IMPORTANT: This notification is *not* guaranteed to be broadcast. For example, if the system goes to sleep because the user removes the batteries, sleep notifications are not sent. Thus, these notifications are unsuitable for applications where external hardware must be shut off to conserve power before the system goes to sleep.

---

sysNotifySleepRequestEvent Specific Data

`notifyDetailsP` points to a `SleepEventParamType` structure.

Prototype

```
typedef struct {
  UInt16 reason;
  UInt16 deferSleep;
} SleepEventParamType;
```

Fields

`reason`
: The reason the system is going to sleep. The possible values are:

`sysSleepAutoOff`
: The idle time limit has been reached.

`sysSleepPowerButton`
: The user pressed the power off button.

`sysSleepResumed`
The sleep event was deferred by one of the notification handlers but has been resumed through the use of the `resumeSleepChr`.

`sysSleepUnknown`
Unknown reason.

`deferSleep`
Initially set to 0. If a notification handler wants to defer sleep, then it should increment this value. When `deferSleep` is greater than 0, the system waits before going to sleep.

Compatibility Implemented only if Notification Feature Set is present.

sysNotifySyncFinishEvent

The `sysNotifySyncFinishEvent` is broadcast immediately after a HotSync operation has completed. Register for this notification if you need to perform post-processing after HotSync operations.

sysNotifySyncFinishEvent Specific Data

None.

Compatibility Implemented only if Notification Feature Set is present.

sysNotifySyncStartEvent

The `sysNotifySyncStartEvent` is broadcast immediately before a HotSync operation is begun. Register for this notification if you need to perform preprocessing before a HotSync operation.

sysNotifySyncStartEvent Specific Data

None.

Compatibility Implemented only if Notification Feature Set is present.

sysNotifyTimeChangeEvent

The `sysNotifyTimeChangeEvent` notification is broadcast just after the system time has been changed using `TimSetSeconds()`. Register for this notification if you need to know when the time has changed.

sysNotifyTimeChangeEvent Specific Data

None.

Compatibility Implemented only if Notification Feature Set is present.

sysNotifyVirtualCharHandlingEvent

The `sysNotifyVirtualCharHandlingEvent` is broadcast to enable custom handling of virtual characters. Set the `handled` field in the `SysNotifyParamType` structure to `true` to prevent the virtual character from being passed on to the rest of the system.

sysNotifyVirtualCharHandlingEvent Specific Data

`notifyDetailsP` points to a `SysNotifyVirtualCharHandlingType` structure.

Prototype

```
typedef struct SysNotifyVirtualCharHandlingTag{
  UInt32 version;
  struct _KeyDownEventType keyDown;
} SysNotifyVirtualCharHandlingType;
```

Fields

`version`
: The current version of this structure. The current version is 0.

`keyDown`
: The virtual character. See the description of the `keyDownEvent` in "Event Reference" on page 45 for a complete description of this structure and its contents.

Compatibility This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don't broadcast this notification. Palm OS Garnet releases do broadcast it. Later versions may or may not broadcast this notification.

sysNotifyVolumeMountedEvent

The `sysNotifyVolumeMountedEvent` is broadcast when a Virtual File System Manager volume is mounted. When a volume is mounted, the VFS Manager activates the `start.prc` application on the newly mounted volume and switches applications to the Launcher or to the `start.prc` application on that volume if it has a user interface.

Register for this notification if you need to know when a volume is mounted or if you want to prevent the default behavior of the VFS Manager.

To prevent the VFS Manager from activating the `start.prc` application, set the `vfsHandledStartPrc` bit in the `handled` field. To prevent the VFS Manager from switching applications, set the `vfsHandledUIAppSwitch` bit.

sysNotifyVolumeMountedEvent Specific Data

`notifyDetailsP` points to a `VFSSlotMountParamType` or `VFSPOSEMountParamType` structure.

Compatibility Implemented only if 4.0 New Feature Set is present.

sysNotifyVolumeUnmountedEvent

The `sysNotifyVolumeUnmountedEvent` is broadcast when a Virtual File System Manager volume is unmounted. Register for this notification if you need to know when a volume is unmounted.

sysNotifyVolumeUnmountedEvent Specific Data

`notifyDetailsP` contains the volume reference number.

Compatibility Implemented only if 4.0 New Feature Set is present.

4

Attention Manager

This chapter provides reference material for the Attention Manager, and is divided into the following sections:

- Attention Manager Data Structures
- Attention Manager Constants
- Attention Manager Functions
- Application-Defined Functions

The Attention Manager API is declared in the header file `AttentionMgr.h`.

For more information about the attention manager, see the section "Getting the User's Attention" in the *Palm OS Programmer's Companion*, vol. I.

IMPORTANT: The Attention Manager was introduced in Palm OS® 4.0 and is not available in earlier versions of the operating system.

Attention Manager Data Structures

AttnCommandType Typedef

Purpose The `AttnCommandType` typedef specifies the set of possible commands that can be sent to the application requesting the user's attention, either as a parameter to the `AttnCallbackProc()` callback function or as one of the arguments that accompanies a `sysAppLaunchCmdAttention` launch code.

Prototype

```
typedef UInt16 AttnCommandType;
```

The following are the values that `AttnCommandType` can assume.

`kAttnCommandDrawDetail`
: Indicates that the application needs to draw the detailed contents of the attention dialog. The command arguments parameter points to a structure of type `drawDetail`.

`kAttnCommandDrawList`
: Indicates that the application needs to draw the appropriate list item in the attention dialog. The command arguments parameter points to a structure of type `drawList`.

`kAttnCommandPlaySound`
: Indicates that the application needs to play a sound. The command arguments parameter is `NULL`.

`kAttnCommandCustomEffect`
: Indicates that the application needs to perform any application-specific special effects. This command is only sent for attention items that set the `kAttnFlagsCustomEffectBit` when they call `AttnGetAttention()`, which most applications won't do.

`kAttnCommandGoThere`
: Tells the application to navigate to the item. The command arguments parameter is `NULL`. An application commonly calls `SysAppLaunch()` upon receipt of this command to have itself launched.

`kAttnCommandGotIt`
: Tells the application that the user is dismissing the item. The command arguments parameter points to a structure of type `gotIt`. The application may choose to clean up memory at this point.

`kAttnCommandSnooze`
: Indicates to the application that the user is snoozing. The command arguments parameter is `NULL`. Most applications do nothing upon receipt of this command. This command is passed to the appropriate application for each and every item currently pending, insistent or subtle. Applications with more than one attention item pending are called more than once.

`kAttnCommandIterate`
: This command is passed to the application during the enumeration of attention items. It is particularly useful after HotSync® operations, as it allows the application to examine

each item, updating or removing those that are stale or invalid. The command arguments parameter points to a structure of type iterate.

AttnCommandArgsType Struct

The AttnCommandArgsType structure is a union of C structures. How you interpret the union's contents depends on which command it accompanies. Not all commands are accompanied by an AttnCommandArgsType structure, as listed in the following table:

AttnCommandType	Accompanied By
kAttnCommandDrawDetail	drawDetail
kAttnCommandDrawList	drawList
kAttnCommandPlaySound	None
kAttnCommandCustomEffect	None
kAttnCommandGoThere	None
kAttnCommandGotIt	gotIt
kAttnCommandSnooze	None
kAttnCommandIterate	iterate

The structures that make up the AttnCommandArgsType union are described in the following sections.

drawDetail

When kAttnCommandDrawDetail is passed to the application, either via the callback function or as a parameter accompanying the sysAppLaunchCmdAttention launch code, the application needs to draw the detailed contents of the attention dialog. The drawDetail structure accompanies the kAttnCommandDrawDetail command, and provides the information needed to draw the contents of that dialog.

Prototype

```
struct AttnCommandArgsDrawDetailTag {
  RectangleType bounds;
  Boolean firstTime;
  AttnFlagsType flags;
} drawDetail;
```

Fields

`bounds`
: Contains the window-relative bounding box for the area to draw. The clipping region is also set to the dimensions of this box to prevent accidental drawing outside.

`firstTime`
: Set to `true` if the user has not yet seen this item. The value of this field could be used to display attentions that the user hasn't seen before in some special way. `firstTime` also indicates to your application whether or not the area in which your application is to draw has been erased. If `firstTime` is `false`, the area is not guaranteed to be blank; your application will need to erase it.

`flags`
: The global user preferences for this attention attempt combined with the custom flags passed in by the developer. Only the lower 16 bits of this field have meaning; use `kAttnFlagsSoundBit`, `kAttnFlagsLEDBit`, `kCustomFlagsVibrateBit`, and `kAttnFlagsCustomEffectBit` (described under "AttnFlagsType" on page 124) to interpret the contents of this field.

drawList

When `kAttnCommandDrawList` is passed to the application, either via the callback function or as a parameter accompanying the `sysAppLaunchCmdAttention` launch code, the application needs to draw the appropriate list item in the attention dialog. The `drawList` structure accompanies the `kAttnCommandDrawList` command, and provides the information needed to draw the contents of that dialog.

Prototype

```
struct AttnCommandArgsDrawListTag {
  RectangleType bounds;
  Boolean firstTime;
  AttnFlagsType flags;
  Boolean selected;
} drawList;
```

Fields

`bounds`
: Contains the window-relative bounding box for the area to draw. The clipping region is also set to the dimensions of this box to prevent accidental drawing outside.

`firstTime`
: Set to `true` if the user has not yet seen this item. The value of this field could be used, for example, to trigger a custom sound the first time this attention item is presented to the user.

`flags`
: The global user preferences for this attention attempt combined with the custom flags passed in by the developer. Only the lower 16 bits of this field have meaning; use `kAttnFlagsSoundBit`, `kAttnFlagsLEDBit`, `kCustomFlagsVibrateBit`, and `kAttnFlagsCustomEffectBit` (described under "AttnFlagsType" on page 124) to interpret the contents of this field.

`selected`
: Set to `true` if the item has been selected. It is up to your code to draw the item appropriately (typically by changing the UI colors) based upon the value of this flag.

gotIt

When `kAttnCommandGotIt` is passed to the application, either via the callback function or as a parameter accompanying the `sysAppLaunchCmdAttention` launch code, it is accompanied by an `gotIt` structure. This structure indicates whether the `kAttnCommandGotIt` command was generated because the user dismissed the attention, or whether the system is simply informing your application that `AttnForgetIt()` was called. Your application normally ignores the latter case if your application made the call to `AttnForgetIt`.

Prototype

```
struct AttnCommandArgsGotItTag {
  Boolean dismissedByUser;
} gotIt;
```

Fields

`dismissedByUser`
: `true` indicates that the user dismissed the attention. `false` indicates that the `kAttnCommandGotIt` command was generated by a call to `AttnForgetIt()`.

iterate

When `kAttnCommandIterate` is passed to the application, either via the callback function or as a parameter accompanying the `sysAppLaunchCmdAttention` launch code, it is accompanied by an `iterate` structure. This structure contains any necessary data that the application may need in order to process the callback or launch code.

Prototype

```
struct AttnCommandArgsIterateTag {
  UInt32 iterationData;
} iterate;
```

Fields

`iterationData`
: Any necessary data that the application may need in order to process the callback or launch code. The value of this field is that which was originally passed to `AttnIterate()`.

AttnFlagsType Typedef

Pass a value of this type to `AttnGetAttention()` and `AttnUpdate()` to specify what means the device should or should not use to get the user's attention. A value of this type is also passed back to your code either as a parameter to the callback function or, if no callback was specified, as part of the structure passed with the `sysAppLaunchCmdAttention` launch code.

Prototype

```
typedef UInt32 AttnFlagsType;
```

Note that more bits may be defined if necessary to accommodate additional hardware.

Constant	Value	Description
`kAttnFlagsSoundBit`	`0x0001`	Plays a sound.
`kAttnFlagsLEDBit`	`0x0002`	Blinks an LED, if the device is so equipped.
`kAttnFlagsVibrateBit`	`0x0004`	Triggers vibration, if the device is so equipped.
`kAttnFlagsCustomEffectBit`	`0x0008`	Triggers an application-specific custom effect.
`kAttnFlagsAllBits`	`0xFFFF`	Uses all available means to get the user's attention.
`kAttnFlagsUseUserSettings`	`0x0000`	System-wide preferences determine what means are used to get the user's attention.

The following constant values can be used to override the user's settings and force or prevent specific behaviors:

Constant	Value	Description
`kAttnFlagsAlwaysSound`	`kAttnFlagsSoundBit`	Play a sound, regardless of the user's settings.
`kAttnFlagsAlwaysLED`	`kAttnFlagsLEDBit`	Blink an LED, if the device is so equipped, regardless of the user's settings.
`kAttnFlagsAlwaysVibrate`	`kAttnFlagsVibrateBit`	Vibrate, if the device is so equipped, regardless of the user's settings.
`kAttnFlagsAlwaysCustomEffect`	`kAttnFlagsCustomEffectBit`	Trigger an application-specific custom effect.
`kAttnFlagsEverything`	`kAttnFlagsAllBits`	Use every available means to get the user's attention, regardless of the user's settings.

Constant	Value	Description
`kAttnFlagsNoSound`	`kAttnFlagsSoundBit << 16`	Prevent a sound from being played, regardless of the user's settings.
`kAttnFlagsNoLED`	`kAttnFlagsLEDBit << 16`	Prevent the LED from flashing, regardless of the user's settings.
`kAttnFlagsNoVibrate`	`kAttnFlagsVibrateBit << 16`	Prevent vibration, regardless of the user's settings.
`kAttnFlagsNoCustomEffect`	`kAttnFlagsCustomEffectBit << 16`	Prevent triggering of the application-specific custom effect.
`kAttnFlagsNothing`	`kAttnFlagsAllBits << 16`	Disable all attention-getting mechanisms, regardless of the user's settings.

These constants can be used in combination. For example, to disable both sound and the LED, use `kAttnFlagsNoSound | kAttnFlagsNoLED`.

If neither `kAttnFlagsAlwaysSound` nor `kAttnFlagsNoSound` is set for a given attention item, a sound plays if and only if the user's preference is to play a sound and the user's preference for alarm volume is non-zero.

AttnLaunchCodeArgsType Struct

If a callback function is not specified in a call to `AttnGetAttention()` and the Attention Manager needs your code to draw the details of your attention in the attention dialog or perform another attention-specific function, it sends a `sysAppLaunchCmdAttention` launch code to your application. Along with the launch code, it passes a pointer to the following

structure, which indicates both what your code is expected to do and identifies the attention that triggered the launch code:

Prototype

```
typedef struct {
  AttnCommandType command;
  UInt32 userData;
  AttnCommandArgsType *commandArgsP;
} AttnLaunchCodeArgsType;
```

Fields

`command`
: Indicates what your code is being requested to do. The complete list of possible commands are described in the definition of `AttnCommandType`.

`userData`
: Identifier that distinguishes the particular attention item from others made by this application. This identifier was specified when the attention item was created.

`commandArgsP`
: Pointer to command-specific arguments. See the description of each command for a discussion of that command's arguments.

When processing the launch code be aware that your application doesn't have application globals available to it; it is important that anything necessary to draw or otherwise display be available through `commandArgsP`.

AttnLevelType Typedef

Attention attempts can either be insistent or subtle. Insistent attention attempts make a serious effort to get the user's attention, by both displaying a dialog and possibly by triggering one or more special effects, such as blinking a light, vibrating, or playing a sound. Other alerts are of a less serious nature and shouldn't disrupt the user. Consequently, subtle attention attempts typically make the attention indicator blink and may trigger one or more special effects, but don't display the Attention Manager dialog. The user can then work until a suitable time, at which point they can tap on the indicator to see what needs their attention. Subtle attention attempts might be used for telling the user that they have new e-mail, or perhaps that a holiday or birthday is coming up.

Prototype `typedef UInt16 AttnLevelType;`

The following are the two defined values for `AttnLevelType`:

`kAttnLevelInsistent`
: An insistent attention attempt. Make a serious effort to get the user's attention by displaying a dialog and optionally triggering one or more special effects.

`kAttnLevelSubtle`
: A subtle attention attempt. Notify the user using special effects, but don't disrupt the user with the dialog if the device is in use.

Note that user preferences for the various special effects can't be set separately for subtle and insistent attention attempts. If your application needs to vary the effects based upon the `AttnLevelType`, pass a suitable value for the `flags` parameter in your `AttnGetAttention()` call.

Attention Manager Constants

In addition to the constant values defined specifically for use with `AttnCommandType`, `AttnFlagsType`, and `AttnLevelType`, the Attention Manager defines the following constant types:

- Error Code Constants
- Attention Manager Drawing Constants
- Attention Manager Feature Constants

Error Code Constants

The Attention Manager returns the following error code under the circumstances described below.

Constant	Description
`attnErrMemory`	Returned by `AttnGetAttention()` when there is insufficient memory to perform the requested operation.

Attention Manager Drawing Constants

The following four constants define the on-screen boundaries of the attention indicator:

Constant	Value	Description
`kAttnIndicatorLeft`	0	The left-hand edge of the attention indicator.
`kAttnIndicatorTop`	0	The top-most edge of the attention indicator.
`kAttnIndicatorWidth`	16	The width of the attention indicator.
`kAttnIndicatorHeight`	16	The height of the attention indicator.

The following two constants are used when drawing the list view. Applications should use these constants to format the display of information in the Attention Manager's list view. Draw the application's small icon centered within the first `kAttnListMaxIconWidth` pixels of the drawing bounds. Then draw two lines of text describing the attention, left-justified, starting at `kAttnListTextOffset` from the left edge of the drawing bounds.

Constant	Value	Description
`kAttnListMaxIconWidth`	15	Maximum width of the application's icon. If the icon is narrower than this, it should be drawn centered within this width.
`kAttnListTextOffset`	17	Offset, from the left-hand edge of the drawing bounds, of the textual description of the attention.

Attention Manager Feature Constants

The Attention Manager defines a read-only feature (`'attn', 0`) that indicates the current user settings and capabilities of the hardware. The upper 16 bits of the feature indicate whether or not the hardware has the capability to perform that sort of alert. The lower 16 bits indicate whether the user has currently enabled that sort of alert.

Constant	Value	Description
`kAttnFtrCreator`	`'attn'`	Attention Manager feature creator.
`kAttnFtrCapabilities`	`0`	Attention Manager feature number.

When working with the value obtained with `FtrGet`, use the following two constants to separate those bits that contain the user settings from those bits that identify the device's capabilities:

Constant	Value	Description
`kAttnFlagsUserSettingsMask`	`kAttnFlagsAllBits`	Mask to isolate those bits that contain the user settings.
`kAttnFlagsCapabilitiesMask`	`kAttnFlagsAllBits << 16`	Mask to isolate those bits that contain the device capabilities.

These constants can be used to interpret the device capabilities (`kAttnFlagsHas...`) and the user settings (`kAttnFlagsUserWants...`):

Constant	Value	Description
`kAttnFlagsHasLED`	`kAttnFlagsLEDBit << 16`	The device has an LED that can be illuminated to indicate an alert.
`kAttnFlagsHasSound`	`kAttnFlagsSoundBit << 16`	The device is capable of playing a sound to indicate an alert.
`kAttnFlagsHasVibrate`	`kAttnFlagsVibrateBit << 16`	The device is capable of vibrating to indicate an alert.

Constant	Value	Description
`kAttnFlagsHasCustomEffect`	`kAttnFlagsCustomEffectBit << 16`	Not used.
`kAttnFlagsUserWantsLED`	`kAttnFlagsLEDBit`	The user wants the LED illuminated to signal an alert.
`kAttnFlagsUserWantsSound`	`kAttnFlagsSoundBit`	The user wants a sound played to signal an alert.
`kAttnFlagsUserWantsVibrate`	`kAttnFlagsVibrateBit`	The user wants the device to vibrate to signal an alert.
`kAttnFlagsUserWantsCustomEffect`	`kAttnFlagsCustomEffectBit`	Not used.

Attention Manager Functions

AttnDoSpecialEffects Function

Purpose Triggers an Attention Manager special effect set.

Declared In `AttentionMgr.h`

Prototype

```
Err AttnDoSpecialEffects (AttnFlagsType flags)
```

Parameters

→ *flags*

Specifies the behavior to be exhibited by this special effects request. See `AttnFlagsType` for the various bits that make up this flag. Note that the behavior is undefined if you set incompatible flags. Supply `kAttnFlagsUseUserSettings` to have this attention request follow the user's pre-set preferences.

Returns Returns `errNone` if no problems were encountered. Returns `attnErrMemory` if there wasn't enough memory to accommodate the attention request.

Comments This routine is provided as a convenience for applications that need to trigger special effects. It does the equivalent of one "nag" of an Attention Manager special effect set.

Compatibility Implemented only if 4.0 New Feature Set is present.

AttnForgetIt Function

Purpose Provides a way for applications to tell the Attention Manager to forget about an attention item.

Declared In `AttentionMgr.h`

Prototype

```
Boolean AttnForgetIt (UInt16 cardNo,
   LocalID dbID,
   UInt32 userData)
```

Parameters

→ *cardNo*
Card number on which the application making the request resides.

→ *dbID*
Database ID of the application making the request.

→ *userData*
Identifier that distinguishes the attention attempt from others made by the same application. This identifier can be an integer, a pointer, or any other 32-bit value.

Returns Returns `true` if the item was removed, `false` if a matching item was not found.

Comments You typically call this function after your application has handled a "Go There" event and the user has read about the item. For example, if there is a subtle attention pending that says "you have three e-mail messages waiting" and you go to the e-mail application on your own and read your e-mail, the subtle notification must disappear. `AttnForgetIt` allows the application to do this.

Note that this call can be made when the Attention Manager dialog is on-screen (though presumably that is rare, since the application is probably not doing much at this point). If this call removes a list item, then the Attention Manager may call back into other items to redraw the list.

If this call removes the last item when any indicator is present, the indicator disappears. If this call removes the last unread item, but read items remain, the indicator switches from blinking to steady state.

Compatibility Implemented only if 4.0 New Feature Set is present.

AttnGetAttention Function

Purpose Requests the user's attention.

Declared In `AttentionMgr.h`

Prototype

```
Err AttnGetAttention (UInt16 cardNo,
   LocalID dbID,
   UInt32 userData,
   AttnCallbackProc *callbackFnP,
   AttnLevelType level, AttnFlagsType flags,
   UInt16 nagRateInSeconds, UInt16
   nagRepeatLimit)
```

Parameters

→ *cardNo*
Card number on which the application making the request resides.

→ *dbID*
Database ID of the application making the request.

→ *userData*
Application-specific data that is later passed back to your code through the callback function. If no callback function is specified in the `callbackFnP` parameter, this data is included in what is passed along with a `sysAppLaunchCmdAttention` launch code. `userData` can be an integer, a pointer, or any other 32-bit value as needed by your application. Most applications pass the unique ID or other key for the record which caused the attention request. `userData` is also used to distinguish a given attention attempt from others made by the same application.

→ *callbackFnP*
Pointer to the function registered by the application to be called by the Attention Manager when the attention is displayed or removed. See `AttnCallbackProc()`, below, for the callback function's parameters. Supply `NULL` to

instead have a `sysAppLaunchCmdAttention` launch code sent to the application that made the attention request whenever the attention is displayed or removed.

→ *`level`*
Indicates the annoyance level. Pass one of the values defined for `AttnLevelType`.

→ *`flags`*
Behavior override, if necessary, for this attention request. This override allows, for instance, silent alarms or noisy alarms. See `AttnFlagsType` for the various bits that make up this flag. Note that the behavior is undefined if you set incompatible flags. Supply `kAttnFlagsUseUserSettings` to have this attention request follow the user's pre-set preferences.

→ *`nagRateInSeconds`*
How long to wait before nagging.

→ *`nagRepeatLimit`*
How many times to nag, excluding the first attempt.

Returns Returns `errNone` if no problems were encountered. Returns `attnErrMemory` if there wasn't enough memory to accommodate the attention request.

Comments The combination of `cardNo`, `dbID` and `userData` uniquely identify an attention-getting attempt. If another call is made to `AttnGetAttention` with identical values for these arguments, an error is reported. To update or delete an existing attention item, pass these same values to `AttnUpdate()` or `AttnForgetIt()`, respectively.

In response to `AttnGetAttention`, the behavior of the operating system or application depends on whether there already are other demands and on the annoyance level passed in the `AttnGetAttention` call.

- No other demands, insistent attention request:

  The Attention Manager puts up a dialog that details the current attempt to get the user's attention.

- Other demands exist, insistent attention request:

  The Attention Manager adds a summary of the current attempt to get the user's attention to a list of things that need

attention. If the dialog is currently in detail form—which is the case if just one other demand exists—the view is refreshed, changing from detail to list form. In this case, the pen and key event queues are also flushed so that any user events that are happening while the display is changing are explicitly ignored. Two exceptions to this behavior exist: if all existing attentions are subtle, or if all existing insistent attentions were snoozed, the new insistent attention brings up the dialog in detail mode, rather than list mode.

- Subtle attention request:

  The Attention Manager starts the attention indicator blinking, and adds the item to its list for later display, unless the dialog is currently being displayed in list mode. In this event, the new subtle attention item simply appears in the list; the indicator does not blink to announce it.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `AttnUpdate()`

AttnGetCounts Function

Purpose Returns the number of attention items that are currently pending.

Declared In `AttentionMgr.h`

Prototype

```
UInt16 AttnGetCounts (UInt16 cardNo, LocalID dbID,
    UInt16 *insistentCountP, UInt16 *subtleCountP)
```

Parameters

→ *cardNo*
: If this value is zero, counts pending attention items from applications on all cards. Otherwise, counts only pending attention items from applications on the specified card.

→ *dbID*
: If this value is zero, counts pending attention items from all applications. Otherwise, counts only pending attention items from applications with the specified database ID.

← *insistentCountP*
: Pointer to a 16-bit unsigned value that is filled in with the number of insistent items pending. Pass `NULL` for this parameter if you don't need to know the number of insistent items that are pending.

← *subtleCountP*
Pointer to a 16-bit unsigned value that is filled in with the number of subtle items pending. Pass `NULL` for this parameter if you don't need to know the number of subtle items that are pending.

Returns Returns the total number of items, both insistent and subtle, that are currently pending.

Comments Call this function if you need to exhibit different behavior if attention items are already pending.

Compatibility Implemented only if 4.0 New Feature Set is present.

AttnIndicatorEnable Function

Purpose Enables and disables the on-screen attention indicator.

Declared In `AttentionMgr.h`

Prototype

```
void AttnIndicatorEnable (Boolean enableIt)
```

Parameters → *enableIt*
`true` to enable the attention indicator, `false` to disable it.

Returns Returns nothing.

Comments This function is used by applications to enable or disable the on-screen attention indicator. The indicator only blinks when all of the following are true:

- The indicator is enabled.
- The indicator is being asked to blink by the attention manager.
- The operating system isn't using the display in such a way as to prevent the attention indicator from showing, such as when the menu bar is being displayed or when a modal dialog is on top of the form.

The attention indicator is enabled by default. If your application controls the upper portion of the screen and needs to prevent the attention indicator from being displayed, call `AttnIndicatorEnable()` and pass it a value of `false`.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `AttnIndicatorEnabled()`

AttnIndicatorEnabled Function

Purpose Returns whether the on-screen attention indicator is currently enabled.

Declared In `AttentionMgr.h`

Prototype

```
Boolean AttnIndicatorEnabled (void)
```

Parameters `None.`

Returns Returns `true` if the on-screen attention indicator is currently enabled, `false` otherwise.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `AttnIndicatorEnable()`

AttnIterate Function

Purpose Instructs the Attention Manager to check each attention item currently pending and, for those that match the specified card number and database ID, invoke the item's callback routine. If a callback routine was not specified in the request, the `sysAppLaunchCmdAttention` launch code is sent to the application that made the attention request.

Declared In `AttentionMgr.h`

Prototype

```
void AttnIterate (UInt16 cardNo, LocalID dbID,
    UInt32 iterationData)
```

Parameters

→ *`cardNo`*
Card number on which the application that made the request resides.

→ *`dbID`*
Database ID of the application that made the request.

→ *`iterationData`*
Any necessary data that the application may need in order to process the callback or launch code. See the description of the

`AttnCallbackProc()` function for more information on this parameter.

Returns Returns nothing.

Comments This function iterates through all of the attention requests made by this application and uses the callback or launch code for each to inform the application about the attention request. When an application receives a `sysAppLaunchCmdSyncNotify` launch code, signifying that a HotSync occurred that affected that application's databases, the application typically calls `AttnIterate` so it can remove attention requests for records that may have been removed during the HotSync. Applications can also call `AttnGetAttention()` after a HotSync, if necessary.

Note that you can call `AttnForgetIt()` inside the iteration since it only marks the record for deletion and thus doesn't confuse the iteration.

Compatibility Implemented only if 4.0 New Feature Set is present.

AttnListOpen Function

Purpose Displays the attention dialog in list mode and, after the user has dismissed it, acts accordingly based on how it was dismissed.

Declared In `AttentionMgr.h`

Prototype

```
void AttnListOpen (void)
```

Parameters `None.`

Returns Returns nothing.

Comments This function allows applications that do not provide the blinking attention indicator to open the list, if necessary.

Compatibility Implemented only if 4.0 New Feature Set is present.

AttnUpdate Function

Purpose Updates one or more aspects of a specified attention item.

Declared In `AttentionMgr.h`

Prototype

```
Boolean AttnUpdate (UInt16 cardNo, LocalID dbID,
   UInt32 userData,
   AttnCallbackProc *callbackFnP,
   AttnFlagsType *flagsP,
   UInt16 *nagRateInSecondsP,
   UInt16 *nagRepeatLimitP)
```

Parameters

→ *cardNo*
Card number on which the application that made the request resides.

→ *dbID*
Database ID of the application that made the request.

→ *userData*
Application-specific data that is passed back to your code through the callback function. If no callback function is specified in the `callbackFnP` parameter, this data is included in what is passed along with a `sysAppLaunchCmdAttention` launch code. `userData` can be an integer, a pointer, or any other 32-bit value. Most applications pass the unique ID or other key for the record which caused the attention request. The value of the `userData` parameter is also used to distinguish a given attention attempt from others made by the same application.

→ *callbackFnP*
Registers a new function to be called by the Attention Manager when the attention is displayed or removed. The function to which this parameter points should conform to `AttnCallbackProc()`. Supply `NULL` to instead have a `sysAppLaunchCmdAttention` launch code sent to the application that made the attention request whenever the attention is displayed or removed.

IMPORTANT: Because `NULL` indicates that a launch code should be sent whenever the callback would otherwise be invoked, it isn't used in this instance to leave the original setting for the `callbackFnP` parameter intact. The value supplied for this parameter *always* overwrites the value supplied in the original attention request.

→ *`flagsP`*
: Pointer to a set of flags that can be used to override user-specified attention behavior; for instance, to force silent or noisy alarms. See `AttnFlagsType` for the various bits that make up this flag, and note that the behavior is undefined if you set incompatible flags. Pass `NULL` to leave the current flag settings unchanged.

→ *`nagRateInSecondsP`*
: Pointer to the length of time to wait before nagging. Pass `NULL` to leave the "nag rate" unchanged.

→ *`nagRepeatLimitP`*
: Pointer to the maximum number of times the user should be nagged. Pass `NULL` to leave the nag repeat limit unchanged.

Returns Returns `true` if the update was successful, `false` if no matching attention item was found.

Comments This call may result in the callback function being called to re-display the item. If no callback function is specified, the `sysAppLaunchCmdAttention` launch code is instead sent to your application. It may also result in callbacks to other pending attention requests.

You call `AttnUpdate` to tell the Attention Manager to update, forcing it to call into all of its clients to redraw. This provides a way for an application to update the text of an attention item without tearing down and re-opening the Attention Manager dialog. For example, `AttnUpdate` could be used to update an existing email notification to say "You have three new email messages" when additional messages are received.

Although `AttnUpdate` may cause a given attention item to redraw, it does not rerun the special effects (if any) that occurred when that attention item was added. If you want to trigger Attention Manager

effects for a particular item, call `AttnForgetIt()` followed by `AttnGetAttention()`.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `AttnGetAttention()`

Application-Defined Functions

AttnCallbackProc Function

Purpose Provides a function prototype to be used by callback functions supplied to `AttnGetAttention()` and `AttnUpdate()`. The supplied function is invoked by the Attention Manager whenever the attention is displayed or removed.

Declared In `AttentionMgr.h`

Prototype

```
typedef Err AttnCallbackProc
    (AttnCommandType command, UInt32 userData,
    AttnCommandArgsType *commandArgsP)
```

Parameters

→ *command*
Indicates what the callback function is being requested to do. The complete list of possible commands are described in the definition of `AttnCommandType`.

→ *userData*
Identifier that distinguishes the particular attention item from others made by this application. This identifier was specified when the attention item was created.

→ *commandArgsP*
Pointer to command-specific arguments. See the description of each command for a discussion of that command's arguments.

Returns The callback function should return `errNone` if it correctly handled the command, or an appropriate error code otherwise. If the callback function returns an error code other than `errNone`, the attention is removed from the list of active attention items.

Comments For a given attention item, the Attention Manager calls back to the code resource that created that item whenever the Attention

Manager needs the resource to draw the attention dialog contents or whenever it needs to inform the code resource of activity relating to the attention item. The Attention Manager calls back using one of two mechanisms:

- If a callback routine has been specified for a given attention item, the Attention Manager invokes the specified routine. This callback routine doesn't have application globals available to it, so it is important that anything necessary to draw or otherwise display be available through `commandArgsP`. A callback routine is typically used by libraries and system extensions.
- If a callback routine has not been specified for a given attention item, the Attention Manager instead sends a `sysAppLaunchCmdAttention` launch code to the application that registered the attention item. Accompanying that launch code is an `AttnLaunchCodeArgsType` structure containing the three parameters documented above. Applications typically use the launch-code mechanism due to the restrictions that are placed on callback routines.

IMPORTANT: It is your responsibility to ensure that the callback procedure is still in the same place when it gets called, dealing with the possibility that the code resource might be unlocked and moved in memory, and with the possibility that the database containing the code resource might be deleted. For the most part, these problems don't exist when using launch codes.

Compatibility Invoked only if 4.0 New Feature Set is present.

5

Categories

This chapter describes the category API as declared in the header file `Category.h`. It discusses the following topics:

- Category Data Structures
- Category Constants
- Category Functions

For more information on categories, see the section "Categories" on page 124 in the *Palm OS Programmer's Companion*, vol. I.

Category Data Structures

AppInfoPtr Typedef

Purpose The `AppInfoPtr` defines a pointer to an `AppInfoType` structure.

Prototype

```
typedef AppInfoType *AppInfoPtr;
```

AppInfoType Struct

Purpose The `AppInfoType` structure shown below maps category names to category indexes and unique IDs. To use the category API described in this chapter, a database's application info block must either be an `AppInfoType` structure, or it must have an `AppInfoType` structure as its first field.

Prototype

```
typedef struct {
  UInt16 renamedCategories;
  Char categoryLabels [dmRecNumCategories]
[dmCategoryLength];
  UInt8 categoryUniqIDs[dmRecNumCategories];
  UInt8 lastUniqID;
  UInt8 padding;
} AppInfoType;
```

Fields

`renamedCategories`
: Used by `CategorySetName()` as a bit field indicating which categories have been renamed. Usually cleared by a conduit.

`categoryLabels`
: An array of strings containing the category names. The maximum size of the array is `dmRecNumCategories`, and the maximum length of each string in the array is `dmCategoryLength`. Both of these constants are defined in `DataMgr.h`.

`categoryUniqIDs`
: Category IDs used for synchronization with the desktop database. Unique IDs generated by the device are between 0 and 127. Unique IDs generated by the desktop computer are between 128 and 255.

`lastUniqID`
: Used for sorting and assigning unique IDs.

Comments

Allocate the application info block in the storage heap and use the `DmSetDatabaseInfo()` function to set the database's application info ID to the local ID of this structure. Then, use the `CategoryInitialize()` function to initialize it with a localized list of strings containing the category names.

Category Constants

The following category constants are defined:

Constant	Value	Description
`categoryHideEditCategory`	10000	Used to suppress the "Edit Categories" item.
`categoryDefaultEditCategoryString`	10001	Used to show the default "Edit Categories" item.

NOTE: These constants look like system resource IDs, but they are not. To use a non-default string for the "Edit Categories" item you pass a resource ID of a string containing your title. If you want to use the default or hide the item, you pass one of these constants. They are within the system resource ID range (that is, they are greater than 10000) so that they don't conflict with any other possible value for that parameter.

Compatibility Both `categoryHideEditCategory` and `categoryDefaultEditCategoryString` are defined only if the 3.5 New Feature Set is present.

Category Functions

CategoryCreateList Function

Purpose Populate a popup list with a database's categories.

Declared In `Category.h`

Prototype

```
void CategoryCreateList (DmOpenRef db,
   ListType *listP, UInt16 currentCategory,
   Boolean showAll, Boolean showUneditables,
   UInt8 numUneditableCategories,
   UInt32 editingStrID, Boolean resizeList)
```

Parameters

→ *db*

Open database containing the category information you want to read.

← *listP*
: Pointer to the `ListType` structure that should display the categories.

→ *currentCategory*
: Index of the category to select. The index is the index into the `categoryLabels` array. The default is to have the "Unfiled" category selected.

→ *showAll*
: `true` to include an "All" list item.

→ *showUneditables*
: `true` to show uneditable categories.

→ *numUneditableCategories*
: The number of categories that the user cannot edit. You should store uneditable categories at the beginning of the `categoryLabels` array. For example, it's common to have an "Unfiled" category at position zero that is not editable. This function displays the uneditable categories at the end of the popup list.

→ *editingStrID*
: The resource ID of a `tSTR` resource to use as the Edit Categories list item. To use the default string ("Edit Categories") pass the constant `categoryDefaultEditCategoryString`.

  If you don't want users to edit categories, pass the `categoryHideEditCategory` constant.

→ *resizeList*
: `true` to resize the list to the number of categories. Set to `true` for popups, `false` otherwise.

Returns Returns nothing.

Comments The "All" item is first in the list (if the `showAll` parameter is `true`), followed by the editable categories in the database and then the categories that cannot be edited. The option to edit categories is last in the list and can be suppressed if desired.

You rarely call this function directly. Instead, most applications use `CategorySelect()`, which calls this function and fully manages the user's selection of a category in the popup list. Use

`CategoryCreateList` only if you want more control over the category popup list.

This function obtains the `db` parameter's `appInfoID`, reads the `AppInfoType` structure at that location, and uses the information in it to initialize the `listP`'s items array with the names of the database's categories. You must have already allocated the structure pointed to by `listP`. `CategoryCreateList` does not display the list; use `LstPopupList()` or `LstDrawList()` to do so.

You must balance a call to `CategoryCreateList` with a call to `CategoryFreeList()`. The `CategoryCreateList` function locks the resources for the category names. It also allocates the `listP` items array. `CategoryFreeList` unlocks all resources locked by `CategoryCreateList` and frees all memory allocated by `CategoryCreateList`.

Compatibility Implemented only if 2.0 New Feature Set is present.

The constants `categoryDefaultEditCategoryString` and `categoryHideEditCategory` are defined only if 3.5 New Feature Set is present. In earlier versions, you can suppress the Edit Categories string by passing 0 for the `editingStrID` parameter, or include the item by passing `categoryEditStrID`.

See Also CategoryCreateListV10()

CategoryCreateListV10 Function

Purpose Read a database's categories and set the category list.

Declared In `Category.h`

Prototype

```
void CategoryCreateListV10 (DmOpenRef db,
    ListType *lst, UInt16 currentCategory,
    Boolean showAll)
```

Parameters

→ *db*
Open database containing the category information you want to read.

← *lst*
Pointer to the `ListType` that should display the categories.

→ *currentCategory*
Index of the category to select. The index is the index into the `categoryLabels` array. The default is to have the "Unfiled" category selected.

→ *showAll*
`true` to include an "All" list item.

Returns Returns nothing.

Compatibility This function corresponds to the Palm OS® 1.0 version of `CategoryCreateList()`. It is obsolete.

CategoryEdit Function

Purpose Event handler for the Edit Categories dialog.

Declared In `Category.h`

Prototype

```
Boolean CategoryEdit (DmOpenRef db,
   UInt16 *category, UInt32 titleStrID,
   UInt8 numUneditableCategories)
```

Parameters

→ *db*
Open database containing the categories to be edited.

← *category*
Upon return, the index of the last category selected before the dialog was closed.

→ *titleStrID*
The resource ID of a `tSTR` resource to use as the dialog's title. To use the default string ("Edit Categories"), pass the constant `categoryDefaultEditCategoryString`.

→ *numUneditableCategories*
The number of categories that the user cannot edit. You should store uneditable categories at the beginning of the `categoryLabels` array. For example, it's common to have an "Unfiled" category at position zero that is not editable.

Returns Returns `true` if any of the following conditions are `true`:

- The current category is renamed.
- The current category is deleted.
- The current category is merged with another category.

Comments You rarely call this function directly. The `CategorySelect()` function calls it when the user chooses the Edit Category list item.

This function both displays the Edit Categories dialog and handles the result of the user actions. It updates the `AppInfoType` structure's list of categories and reassigns database records to new categories as needed. If a user deletes a category, `CategoryEdit` moves all of the records belonging to that category to the Unfiled category. If a category is renamed to be the same as an existing category, this function moves all of the old category's records to the new category.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `CategoryEditV20()`, `CategoryEditV10()`, `DmMoveCategory()`

CategoryEditV20 Function

Purpose Event handler for the Edit Categories dialog.

Declared In `Category.h`

Prototype

```
Boolean CategoryEditV20 (DmOpenRef db,
   UInt16 *category, UInt32 titleStrID)
```

Parameters

→ *db*
Database containing the categories to be edited.

← *category*
Upon return, the last category selected before the dialog was closed.

→ *titleStrID*
The resource ID of a `tSTR` resource to use as the dialog's title.

Returns Returns `true` if any of the following conditions are `true`:

- The current category is renamed.
- The current category is deleted.
- The current category is merged with another category.

Compatibility This function corresponds to the Palm OS 2.0 version of `CategoryEdit`. Implemented only if 2.0 New Feature Set is present. This function is obsolete.

See Also CategoryEdit(), CategoryEditV10()

CategoryEditV10 Function

Purpose Event handler for the Edit Categories dialog.

Declared In `Category.h`

Prototype

```
Boolean CategoryEditV10 (DmOpenRef db,
   UInt16 *category)
```

Parameters

→ *db*
Open database containing the categories to be edited.

← *category*
Upon return, the last category selected before the dialog was closed.

Returns Returns `true` if any of the following conditions are true:

- The current category is renamed.
- The current category is deleted.
- The current category is merged with another category.

Compatibility This function corresponds to the Palm OS 1.0 version of `CategoryEdit`. It is obsolete.

See Also CategoryEdit(), CategoryEditV20()

CategoryFind Function

Purpose Return the index of a category given its name.

Declared In `Category.h`

Prototype

```
UInt16 CategoryFind (DmOpenRef db,
   const Char *name)
```

Parameters

→ *db*
Open database to search.

→ *name*
Category name. Pass the empty string to find the first unused category.

Returns Returns the index of the category's entry in the `categoryLabels` array (see `AppInfoType`). Returns `dmAllCategories` if the category does not exist.

CategoryFreeList Function

Purpose Unlock or free memory locked or allocated by `CategoryCreateList()`.

Declared In `Category.h`

Prototype

```
void CategoryFreeList (DmOpenRef db,
   ListType *listP, Boolean showAll,
   UInt32 editingStrID)
```

Parameters

→ *db*
Open database containing the categories.

→ *listP*
Pointer to the category list. (See `ListType`.)

→ *showAll*
`true` if the list was created with an "All" category.

→ *editingStrID*
The resource ID that you passed as the `editingStrID` parameter to `CategoryCreateList()`. This function unlocks the resource.

Returns Returns nothing.

Comments You only need to call this function if you explicitly call `CategoryCreateList()`. Typical applications call `CategorySelect`, which handles both the creation and deletion of the list.

This function frees the items in the popup list `listP`'s items array and it unlocks other resources that `CategoryCreateList` may have locked.

This function does *not* remove the categories from the passed database, and it does *not* free the `ListType` structure pointed to by `listP`. (Typically, a list is freed when its form is freed.)

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `CategoryFreeListV10()`

CategoryFreeListV10 Function

Purpose Unlock or free memory locked or allocated by `CategoryCreateListV10()`.

Declared In `Category.h`

Prototype

```
void CategoryFreeListV10 (DmOpenRef db,
    ListType *lst)
```

Parameters

→ *db*
Open database containing the categories.

→ *listP*
Pointer to the category list. (See `ListType`.)

Returns Returns nothing.

Compatibility This function corresponds to the Palm OS 1.0 version of `CategoryFreeList`. It is obsolete.

See Also CategoryFreeList()

CategoryGetName Function

Purpose Return the name of the specified category.

Declared In `Category.h`

Prototype

```
void CategoryGetName (DmOpenRef db, UInt16 index,
    Char *name)
```

Parameters

→ *db*
Database that contains the categories.

→ *index*
Category index. This is the index into the `categoryLabels` array in the `AppInfoType` structure. You can retrieve this index from a database record's attribute word.

← *name*
Buffer to hold category name. Buffer should be `dmCategoryLength` in size.

Returns Stores the category name in the `name` buffer passed.

May display a fatal error message if the index is out of range.

Comments You can use this function to find out the name of a given database record's category. Use the `DmRecordInfo()` call to obtain the category index from the given record. For example:

```
DmOpenRef myDB;
UInt16 record, attr, category;
Char *name;

DmRecordInfo(myDB, record, &attr, NULL, NULL);
category = attr & dmRecAttrCategoryMask;
CategoryGetName(myDB, category, name);
```

The category's name is copied into the variable you pass for the `name` parameter.

See Also `CategorySetName()`

CategoryGetNext Function

Purpose Return the index of the next category after a given category.

Declared In `Category.h`

Prototype

```
UInt16 CategoryGetNext (DmOpenRef db,
    UInt16 index)
```

Parameters

→ *db*
Open database containing the categories.

→ *index*
Category index.

Returns Category index of next category.

Comments The intended use of this function is to allow your users to cycle through categories. For example, the built-in applications cycle through categories when the user presses the corresponding hard-key button. (See the `ListViewNextCategory` function in the

Address Book sample application for an example.) Note that categories are not displayed in the same order as they are stored.

Do not use this function for searching for a particular category or iterating through a category list.

Compatibility In Palm OS 1.0, the system chose `Unfiled` as one category.

In Palm OS 2.0 and later, the system skips both `Unfiled` and categories with empty records.

CategoryInitialize Function

Purpose Initialize the category names, IDs, and flags.

Declared In `Category.h`

Prototype

```
void CategoryInitialize (AppInfoPtr appInfoP,
   UInt16 localizedAppInfoStrID)
```

Parameters

→ *appInfoP*
Pointer to the locked application info block. See `AppInfoType`.

→ *localizedAppInfoStrID*
Resource ID of the localized category names. This must be a resource of the type `appInfoStringsRsc` (`'tAIS'`).

Returns Returns nothing.

Comments Call this function at database creation time to initialize the database's categories from a list of localized strings.

`CategoryInitialize` initializes the `AppInfoType` structure that is associated with your database. It does not create the structure. To create the structure, you must allocate it in the storage heap (using `DmNewHandle()`) and associate it with your database using `DmSetDatabaseInfo()`.

Compatibility Implemented only if 2.0 New Feature Set is present.

CategorySelect Function

Purpose Process the selection and editing of categories.

Declared In `Category.h`

Prototype

```
Boolean CategorySelect (DmOpenRef db,
   const FormType *frm, UInt16 ctlID,
   UInt16 lstID,
   Boolean title, UInt16 *categoryP,
   Char *categoryName,
   UInt8 numUneditableCategories,
   UInt32 editingStrID)
```

Parameters

→ *db*
Open database containing the categories.

→ *frm*
Form that contains the category popup list.

→ *ctlID*
ID of the popup trigger.

→ *lstID*
ID of the popup list.

→ *title*
`true` to have an "All" list item. (In general, if the trigger is in the form's title bar, it should have an "All" item. If the trigger is elsewhere in the form, it should not.)

↔ *categoryP*
Index of the selected category. The index is the index into the `categoryLabels` array.

↔ *categoryName*
Name of the selected category.

→ *numUneditableCategories*
The number of categories that the user cannot edit. You should store uneditable categories at the beginning of the `categoryLabels` array. For example, it's common to have an "Unfiled" category at position zero that is not editable. This function displays the uneditable categories at the end of the popup list.

→ *editingStrID*
The resource ID of a `tSTR` resource to use as the Edit Categories list item. To use the default string ("Edit

Categories"), pass the constant `categoryDefaultEditCategoryString`.

If you don't want users to edit categories, pass the `categoryHideEditCategory` constant.

Returns Returns `true` if any of the following conditions are true:

- The current category is renamed.
- The current category is deleted.
- The current category is merged with another category.

Comments Call this function when the user taps the category popup trigger. This function handles all aspects of displaying the popup list and managing the user selection—It creates the popup list using `CategoryCreateList()`, displays the popup list, calls `CategoryEdit()` if the user selects the Edit Categories item, uses `CategorySetTriggerLabel()` to set the trigger label to the item the user selected, and then calls `CategoryFreeList()` to free the list items array. Your application is responsible for checking the value of `categoryP` upon return and updating the display or changing the record's category to the new selection.

Compatibility Implemented only if 2.0 New Feature Set is present.

The constants `categoryDefaultEditCategoryString` and `categoryHideEditCategory` are defined only if 3.5 New Feature Set is present. In earlier versions, you can suppress the Edit Categories string by passing 0 for the `editingStrID` parameter, or include the item by passing `categoryEditStrID`.

See Also CategorySelectV10()

CategorySelectV10 Function

Purpose Process the selection and editing of categories.

Declared In `Category.h`

Prototype

```
Boolean CategorySelectV10 (DmOpenRef db,
   const FormType *frm, UInt16 ctlID,
   UInt16 lstID,
   Boolean title, UInt16 *categoryP,
   Char *categoryName)
```

Parameters

→ *db*
Open database containing the categories.

→ *frm*
Form that contains the category popup list.

→ *ctlID*
ID of the popup trigger.

→ *lstID*
ID of the popup list.

→ *title*
`true` to have an "All" list item. (In general, if the trigger is in the form's title bar, it should have an "All" item. If the trigger is elsewhere in the form, it should not.)

↔ *categoryP*
Index of the selected category. The index is the index into the `categoryLabels` array.

↔ *categoryName*
Name of the selected category.

Returns Returns `true` if any of the following conditions are true:

- The current category is renamed.
- The current category is deleted.
- The current category is merged with another category.

Compatibility This function corresponds to the Palm OS 1.0 version of `CategorySelect()`. It is obsolete.

CategorySetName Function

Purpose Change the category name in the `AppInfoType` structure, or delete a category.

Declared In `Category.h`

Prototype

```
void CategorySetName (DmOpenRef db, UInt16 index,
    const Char *nameP)
```

Parameters

→ *db*
Open database containing the category to change.

→ *index*
Index of category to rename.

→ *nameP*
The new category name (null-terminated), or `NULL` to delete the category.

Returns Returns nothing.

Comments The `CategoryEdit()` function calls this function when a user creates a new category or renames an existing category in the Edit Categories dialog. Your application does not have to call it directly.

Compatibility Implemented only if 2.0 New Feature Set is present.

CategorySetTriggerLabel Function

Purpose Set the label displayed by the category popup trigger.

Declared In `Category.h`

Prototype

```
void CategorySetTriggerLabel (ControlType *ctl,
    Char *name)
```

Parameters

↔ *ctl*
Pointer to control object (popup trigger) to relabel.

↔ *name*
Pointer to the name of the new category.

Returns Returns nothing.

Comments The `CategorySetTriggerLabel` function calls the `CategoryTruncateName()` function to truncate the category name to the maximum length. The maximum length varies, depending upon which ROM is installed in the device.

NOTE: This function passes the `name` parameter to the `CategoryTruncateName` function, which means that the `name` value must be modifiable. `CategorySetTriggerLabel` does not make a copy of the string passed, so you must ensure that the string remains valid until the form is closed.

See Also `CtlSetLabel()`

CategoryTruncateName Function

Purpose Truncate a category name so that it's short enough to display. The category name is truncated if it's longer than `maxWidth`.

Declared In `Category.h`

Prototype

```
void CategoryTruncateName (Char *name,
    UInt16 maxWidth)
```

Parameters

↔ *name*
Category name to truncate. Upon return, contains the truncated name.

→ *maxWidth*
Maximum size, in pixels, of truncated category (including ellipsis).

Returns Returns nothing.

6

Clipboard

This chapter provides reference material for the clipboard API defined in `Clipboard.h`. It covers:

- Clipboard Data Structures
- Clipboard Functions

Clipboard Data Structures

ClipboardFormatType Enum

Purpose

The `ClipboardFormatType` enum specifies the type of data to add to the clipboard or retrieve from the clipboard.

Prototype

```
enum clipboardFormats {
  clipboardText,
  clipboardInk,
  clipboardBitmap };
typedef enum clipboardFormats
ClipboardFormatType;
```

Constants

`clipboardText`
: Textual data. This is the most commonly used clipboard.

`clipboardInk`
: Reserved.

`clipboardBitmap`
: Bitmap data.

Comments

Clipboards for each type of data are separately maintained. That is, if you add a string of text to the clipboard, then add a bitmap, then ask to retrieve a `clipboardText` item from the clipboard, you will receive the string you added before the bitmap; the bitmap does not overwrite textual data and vice versa.

Clipboard Functions

ClipboardAddItem Function

Purpose Add the item passed to the specified clipboard. Replaces the current item (if any) of that type.

Declared In `Clipboard.h`

Prototype

```
void ClipboardAddItem
    (const ClipboardFormatType format,
    const void *ptr, UInt16 length)
```

Parameters

→ *format*
Text, ink, bitmap, etc. See `ClipboardFormatType`.

→ *ptr*
Pointer to the item to place on the clipboard.

→ *length*
Size in bytes of the item to place on the clipboard.

Returns Returns nothing.

Comments The clipboard makes a copy of the data that you pass to this function. Thus, you may free any data that you've passed to the clipboard without destroying the contents of the clipboard. You may also add constant data or stack-based data to the clipboard.

WARNING! You can't add null-terminated strings to the clipboard.

See Also `FldCut()`, `FldCopy()`

ClipboardAppendItem Function

Purpose Append data to the item on the clipboard.

Declared In `Clipboard.h`

Prototype

```
Err ClipboardAppendItem
    (const ClipboardFormatType format,
    const void *ptr, UInt16 length)
```

Parameters

→ *format*
Text, ink, bitmap, etc. See `ClipboardFormatType`. This function is intended to be used only for the `clipboardText` format.

→ *ptr*
Pointer to the data to append to the item on the clipboard.

→ *length*
Size in bytes of the data to append to the clipboard.

Returns 0 upon success or `memErrNotEnoughSpace` if there is not enough space to append the data to the clipboard.

Comments This function differs from `ClipboardAddItem()` in that it does not overwrite data already on the clipboard. It allows you to create a large text item on the clipboard from several small disjointed pieces. When other applications retrieve the text from the clipboard, it's retrieved as a single unit.

This function simply appends the specified item to the item already on the clipboard without attempting to parse the format. It's assumed that you'll call it several times over a relatively short interval and that no other application will attempt to retrieve text from the clipboard before your application is finished appending.

Compatibility Implemented only if 3.2 New Feature Set is present.

ClipboardGetItem Function

Purpose Return the handle of the contents of the clipboard of a specified type and the length of a clipboard item.

Declared In `Clipboard.h`

Prototype

```
MemHandle ClipboardGetItem
    (const ClipboardFormatType format,
    UInt16 *length)
```

Parameters

→ *format*
Text, ink, bitmap, etc. See `ClipboardFormatType`.

← *length*
The length in bytes of the clipboard item is returned here.

Returns Handle of the clipboard item.

Comments The handle returned is a handle to the actual clipboard chunk. It is not suitable for passing to any API that modifies memory (such as `FldSetTextHandle()`). Consider this to be read-only access to the chunk. Copy the contents of the clipboard to your application's own storage as soon as possible and use that reference instead of the handle returned by this function.

Don't free the handle returned by this function; it is freed when a new item is added to the clipboard.

Text retrieved from the clipboard does not have a null terminator. You must use the `length` parameter to determine the length in bytes of the string you've retrieved.

7

Controls

This chapter describes the control object API as declared in the header file `Control.h`. It discusses the following topics:

- Control Data Structures
- Control Resources
- Control Functions

For more information on controls, see the section "Offscreen Windows" in the *Palm OS Programmer's Companion*, vol. I.

Control Data Structures

ButtonFrameType Enum

Purpose

The `ButtonFrameType` enum specifies the border style for the button. It defines values for the `frame` field of `ControlAttrType`.

Prototype

```
enum buttonFrames {noButtonFrame,
   standardButtonFrame, boldButtonFrame,
   rectangleButtonFrame};
typedef enum buttonFrames ButtonFrameType;
```

Constants

`noButtonFrame`
: The button has no border.

`standardButtonFrame`
: Standard button rectangular border with rounded corners.

`boldButtonFrame`
: Bolded rectangular border with rounded corners.

`rectangleButtonFrame`
: Rectangular border with square corners.

ControlAttrType Struct

Purpose The `ControlAttrType` bit field specifies the control's visible characteristics. It is defined as follows:

Prototype

```
typedef struct {
  UInt8 usable:1;
  UInt8 enabled:1;
  UInt8 visible:1;
  UInt8 on:1;
  UInt8 leftAnchor:1;
  UInt8 frame:3;
  UInt8 drawnAsSelected:1;
  UInt8 graphical:1;
  UInt8 vertical:1;
} ControlAttrType;
```

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `ControlAttrType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Your code should treat the `ControlAttrType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

Fields

`usable`
: If 0, the control is not considered to be part of the interface of the current application, and it doesn't appear on screen. You can use `CtlSetUsable()`, `CtlShowControl()`, or `CtlHideControl()` to set or clear this value.

`enabled`
: If 0, the control is visible but doesn't respond to the pen. This value is set by `CtlSetEnabled()` and returned by `CtlEnabled()`.

`visible`
: Set and cleared internally when the control is drawn (`CtlDrawControl()`) and erased (`CtlEraseControl()`).

`on`
: If set, the control has the value "on." For example, a check box that has the on value has a check mark displayed in it. Use `CtlGetValue()` and `CtlSetValue()` to retrieve and set this value.

`leftAnchor`
: Used by controls that expand and shrink their width when the label is changed. If this attribute is set, the left bound of the control is fixed.

`frame`
: The type of frame drawn around the button controls. See `ButtonFrameType` for possible values. Only button controls use this attribute; for all other controls, the `ControlStyleType` determines the frame.

`drawnAsSelected`
: Used on Palm OS® release 3.5 for button controls that contain no text (indicating that the button is displayed on top of a bitmap). If set, the button is drawn as inverted. If clear, the button is drawn normally.

`graphical`
: If set, the control is a graphical control, slider, or feedback slider.

`vertical`
: Not currently used.

Compatibility The `drawnAsSelected`, `graphical`, and `vertical` attributes are only present if 3.5 New Feature Set is present.

ControlPtr Typedef

Purpose The `ControlPtr` is a pointer to a `ControlType` structure.

Prototype

```
typedef ControlType *ControlPtr;
```

ControlStyleType Enum

Purpose The `ControlStyleType` enum specifies values for the `ControlType` `style` field, which specifies the type of the control (button, push button, and so on).

```
enum controlStyles {buttonCtl, pushButtonCtl,
   checkboxCtl, popupTriggerCtl,
   selectorTriggerCtl, repeatingButtonCtl,
   sliderCtl, feedbackSliderCtl};
typedef enum controlStyles ControlStyleType;
```

Constants

`buttonCtl`
: Button. Buttons display a text label in a box. The `ButtonFrameType` specifies the type of box.

`pushButtonCtl`
: Push button. Selecting a push button inverts its display so that it appears highlighted.

`checkboxCtl`
: Check box. Check boxes display a setting of either on (checked) or off (unchecked)

`popupTriggerCtl`
: Popup trigger. Popup triggers display a graphic element followed by a text label. They are used to display popup lists.

`selectorTriggerCtl`
: Selector trigger. Selector triggers display a text label surrounded by a gray rectangular frame. The control expands or contracts to the width of the new label.

`repeatingButtonCtl`
: Repeating button. Repeating buttons look like buttons; however, a repeating button is repeatedly selected if the user holds the pen on it.

`sliderCtl`
: Slider. Sliders display two bitmaps: one representing the current value (the thumb), and another representing the scale of available values. The user can slide the thumb to the left or the right to change the value.

`feedbackSliderCtl`
: Feedback slider. A feedback slider looks like a slider; however, a feedback slider sends events each time the thumb moves while the pen is still down. A regular slider sends an event only when the user releases the pen.

Compatibility

The `sliderCtl` and `feedbackSliderCtl` values are only defined if 3.5 New Feature Set is present.

ControlType Struct

Purpose The `ControlType` structure defines the type and characteristics of a control. It is defined as follows:

Prototype

```
typedef struct {
  UInt16 id;
  RectangleType bounds;
  Char *text;
  ControlAttrType attr;
  ControlStyleType style;
  FontID font;
  UInt8 group;
  UInt8 reserved;
} ControlType;
```

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `ControlType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Your code should treat the `ControlType` structure as opaque. The fields in the struct are set by values you specify when you create the control resource, and they typically do not change. Use the functions specified in the descriptions below to retrieve and set the values. Do not attempt to change structure member values directly.

Fields

`id`
: ID value you specified when you created the control resource.

`bounds`
: Bounds of the control, in window-relative coordinates. The control's text label is clipped to the control's bounds. The control's frame is drawn around (outside) the bounds of the control. `FrmGetObjectBounds()` and `FrmSetObjectBounds()` retrieve and set this value.

`text`
: Pointer to the control's label. If `text` is `NULL`, the control has no label. Use `CtlGetLabel()` and `CtlSetLabel()` to retrieve and set this value.

`attr`
: Control attributes. See `ControlAttrType`.

`style`
: Style of the control. See `ControlStyleType`.

`font`
: Font to use to draw the control's label.

`group`
: Group ID of a push button or a check box that is part of an exclusive group. The control routines don't automatically turn one control off when another is selected. It's up to the application or a higher-level object, like a dialog box, to manage this.

`reserved`
: Reserved for future use.

GraphicControlType Struct

Purpose

The `GraphicControlType` struct defines a graphical control. A graphical control is like any other control except that it displays a bitmap in place of the text label.

Prototype

```
typedef struct GraphicControlType {
  UInt16 id;
  RectangleType bounds;
  DmResID bitmapID;
  DmResID selectedBitmapID;
  ControlAttrType attr;
  ControlStyleType style;
  FontID unused;
  UInt8 group;
  UInt8 reserved;
} GraphicControlType;
```

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `GraphicControlType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Your code should treat the `GraphicControlType` structure as opaque. The fields in the struct are set by values you specify when you create the control resource, and they typically do not change. Use the functions specified in the descriptions below to retrieve and set the values. Do not attempt to change structure member values directly.

Fields

`id`
: ID value you specified when you created the control resource.

`bounds`
: Bounds of the control, in window-relative coordinates. The control's frame is drawn around (outside) the bounds of the control. `FrmGetObjectBounds()` and `FrmSetObjectBounds()` retrieve and set this value.

`bitmapID`
: Resource ID of the bitmap to display in the button. You can use `CtlSetGraphics()` to change this value.

`selectedBitmapID`
: If the button should show a different bitmap when selected, this field contains the resource ID of that bitmap. You typically use this field for push buttons or repeating buttons. `CtlSetGraphics()` can change this value.

`attr`
: Control attributes. See `ControlAttrType`. For a graphical control, the `graphical` attribute must be set. The APIs described in the `ControlAttrType` section can be used to access the bitfields here. Because the `ControlAttrType` APIs take a `ControlType*` as an argument, the `GraphicControlType*` should be cast to a `ControlType*` when making the API calls.

`style`
: Style of the control. See `ControlStyleType`. A graphical control can be any type of control other than `checkboxCtl`.

`unused`
: Unused.

`group`
: Group ID of a push button that is part of an exclusive group. The control routines don't automatically turn one control off

when another is selected. It's up to the application or a higher-level object, like a dialog box, to manage this.

`reserved`
Reserved for future use.

Compatibility This struct is defined only if 3.5 New Feature Set is present.

SliderControlType Struct

Purpose The `SliderControlType` struct defines a slider control or a feedback slider control.

Prototype

```
typedef struct SliderControlType {
  UInt16 id;
  RectangleType bounds;
  DmResID thumbID;
  DmResID backgroundID;
  ControlAttrType attr;
  ControlStyleType style;
  UInt8 reserved;
  Int16 minValue;
  Int16 maxValue;
  Int16 pageSize;
  Int16 value;
  MemPtr activeSliderP;
} SliderControlType;
```

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `SliderControlType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Your code should treat the `SliderControlType` structure as opaque. The fields in the struct are set by values you specify when you create the control resource, and they typically do not change. You can use `CtlSetSliderValues()` to set new minimum, maximum, page size, and current values, and `CtlGetSliderValues()` to retrieve these values. Do not attempt to change structure member values directly.

Fields

`id`
ID value you specified when you created the control resource.

`bounds`
Bounds of the control, in window-relative coordinates. `FrmGetObjectBounds()` and `FrmSetObjectBounds()` retrieve and set this value.

`thumbID`
Resource ID of the bitmap to use for the slider knob (called the "thumb"). If `NULL`, the default bitmap is used.

`backgroundID`
Resource ID of the bitmap to use for the slider background. If `NULL`, the default bitmap is used.

`attr`
Control attributes. See `ControlAttrType`. For a slider, the `graphical` attribute is set. The APIs described in the `ControlAttrType` section can be used to access the bitfields here. Because the `ControlAttrType` APIs take a `ControlType*` as an argument, the `SliderControlType*` should be cast to a `ControlType*` when making the API calls.

`style`
Style of the control. See `ControlStyleType`. Must be `sliderCtl` or `feedbackSliderCtl`.

`reserved`
Reserved for future use.

`minValue`
Value of the slider when the thumb is all the way to the left.

`maxValue`
Value of the slider when the thumb is all the way to the right.

`pageSize`
Amount by which to increase or decrease the slider value when the user taps to the right or left of the thumb.

`value`
Current value represented by the slider. Use `CtlGetValue()` and `CtlSetValue()` to retrieve and set this value.

`activeSliderP`
: Pointer to a memory location used when the slider is active. A slider is active if it is currently being drawn or if it is tracking the pen. If the slider is inactive, this pointer is `NULL`.

Compatibility This struct is defined only if 3.5 New Feature Set is present.

Control Resources

Different resources are associated with different controls, as follows:

- Button—Button Resource (tBTN)
- Popup trigger— Popup Trigger Resource (tPUT)
- Selector trigger—Selector Trigger Resource (tSLT)
- Repeat control—Repeating Button Resource (tREP)
- Push button—Push Button Resource (tPBN)
- Check box—Check Box Resource (tCBX)
- Slider— Slider Resource (tsld)
- Feedback slider— Feedback Slider Resource (tslf)

See *Palm OS User Interface Guidelines* for more information.

Control Functions

CtlDrawControl Function

Purpose Draw a control object (and the text or graphic in it) on screen.

Declared In `Control.h`

Prototype
```
void CtlDrawControl (ControlType *controlP)
```

Parameters → *`controlP`*
: Pointer to the control object to draw. (See `ControlType`.)

Returns Returns nothing.

Comments The control is drawn only if its `usable` attribute is `true`. This function sets the `visible` attribute to `true`.

Compatibility In releases prior to Palm OS® 3.5, it is common to create graphical buttons by drawing a button with no text label on top of a bitmap. If 3.5 New Feature Set is present, you should use graphical controls instead. (See `GraphicControlType`.) `CtlDrawControl` attempts to provide backward compatibility for the old-style graphical buttons.

See Also `CtlSetUsable()`, `CtlShowControl()`

CtlEnabled Function

Purpose Return `true` if the control responds to the pen.

Declared In `Control.h`

Prototype

```
Boolean CtlEnabled (const ControlType *controlP)
```

Parameters → *controlP*
Pointer to control object. (See `ControlType`.)

Returns Returns `true` if the controls object responds to the pen; `false` if not.

Comments This function provides no indication of whether the control is visible on the screen. A control that doesn't respond to the pen may be visible, and if so, its appearance is no different from controls that do respond to the pen. You might use such a control to display some state of your application that cannot be modified.

See Also CtlSetEnabled()

CtlEraseControl Function

Purpose Erase a usable and visible control object and its frame from the screen.

Declared In `Control.h`

Prototype

```
void CtlEraseControl (ControlType *controlP)
```

Parameters → *controlP*
Pointer to control object to erase. (See `ControlType`.)

Comments This function sets the `visible` attribute to `false`. If 3.5 New Feature Set is present, it also sets the `drawnAsSelected` attribute to `false`.

Don't call this function directly; instead, use `FrmHideObject()`, which calls this function.

CtlGetLabel Function

Purpose Return a character pointer to a control's text label.

Declared In `Control.h`

Prototype

```
const Char *CtlGetLabel
    (const ControlType *controlP)
```

Parameters → *controlP*
Pointer to control object. (See `ControlType`.)

Returns Returns a pointer to a null-terminated string.

Comments Make sure that `controlP` is not a graphical control or a slider control. The graphical control and slider control structures do not contain a text label field.

See Also `CtlSetLabel()`

CtlGetSliderValues Function

Purpose Return current values used by a slider control.

Declared In `Control.h`

Prototype

```
void CtlGetSliderValues (const ControlType *ctlP,
    UInt16 *minValueP, UInt16 *maxValueP,
    UInt16 *pageSizeP, UInt16 *valueP)
```

Parameters → *ctlP*
Pointer to a control object. (See `ControlType`.)

← *minValueP*
The slider's minimum value. Pass `NULL` if you don't want to retrieve this value.

← *maxValueP*
The slider's maximum value. Pass `NULL` if you don't want to retrieve this value.

← *pageSizeP*
The slider's page size value. Pass `NULL` if you don't want to retrieve this value.

← *valueP*
The slider's current value. Pass `NULL` if you don't want to retrieve this value.

Returns Returns nothing. The slider's values are returned in the parameters to this function.

Comments If `ctlP` is not a slider or a feedback slider, this function immediately returns.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `CtlSetSliderValues()`, `SliderControlType`

CtlGetValue Function

Purpose Return the current value of the specified control.

Declared In `Control.h`

Prototype

```
Int16 CtlGetValue (const ControlType *controlP)
```

Parameters

→ *controlP*
Pointer to a control object. (See `ControlType`.)

Returns Returns the current value of the control. For most controls the return value is either 0 (off) or 1 (on). For sliders, this function returns the value of the `value` field.

See Also `CtlSetValue()`, `FrmGetControlGroupSelection()`, `FrmSetControlGroupSelection()`, `FrmGetControlValue()`, `FrmSetControlValue()`

CtlHandleEvent Function

Purpose Handle event in the specified control object.

Declared In `Control.h`

Prototype

```
Boolean CtlHandleEvent (ControlType *controlP,
    EventType *pEvent)
```

Parameters

→ *controlP*
Pointer to control object. (See `ControlType`.)

→ *pEvent*
Pointer to an EventType structure.

Returns Returns `true` if an event is handled by this function. Events that are handled are:

- `penDownEvent` — If the pen is within the bounds of the control
- `ctlEnterEvent`, `ctlRepeatEvent`, and `ctlExitEvent`— If the control ID in the event data matches the control's ID.

Comments The control object must be usable, visible, and respond to the pen for this function to handle the event.

When this routine receives a `penDownEvent`, it checks if the pen position is within the bounds of the control object. If it is, a `ctlEnterEvent` is added to the event queue and the routine exits.

When this routine receives a `ctlEnterEvent`, the control object is redrawn as necessary as either selected or deselected, depending on its previous state.

When this routine receives a `ctlEnterEvent` or `ctlRepeatEvent`, it checks that the control ID in the passed event record matches the ID of the specified control. If they match, this routine tracks the pen until it comes up or until it leaves the object's bounds. When that happens, `ctlSelectEvent` is sent to the event queue if the pen came up in the bounds of the control. If the pen exits the bounds, a `ctlExitEvent` is sent to the event queue.

CtlHideControl Function

Purpose Set a control's `usable` attribute to `false` and erase the control from the screen.

Declared In `Control.h`

Prototype
```
void CtlHideControl (ControlType *controlP)
```

Parameters → *`controlP`*
Pointer to the control object to hide. (See `ControlType`.)

Returns Returns nothing.

Comments A control that is not usable doesn't draw and doesn't respond to the pen.

This function is the same as `CtlEraseControl()` except that it also sets `usable` to `false` (in addition to setting `visible` to `false`).

Don't call this function directly; instead, use `FrmHideObject()`, which performs the same function and works for all user interface objects.

See Also `CtlShowControl()`

CtlHitControl Function

Purpose Simulate tapping a control. This function adds a `ctlSelectEvent` to the event queue.

Declared In `Control.h`

Prototype
```
void CtlHitControl (const ControlType *controlP)
```

Parameters → *`controlP`*
Pointer to a control object. (See `ControlType`.)

Returns Returns nothing.

Comments Useful for testing.

CtlNewControl Function

Purpose Create a new control object dynamically and install it in the specified form.

Declared In `Control.h`

Prototype

```
ControlType *CtlNewControl (void **formPP,
    UInt16 ID, ControlStyleType style,
    const Char *textP, Coord x, Coord y,
    Coord width,
    Coord height, FontID font, UInt8 group,
    Boolean leftAnchor)
```

Parameters

↔ *formPP*
: Pointer to the pointer to the form in which the new control is installed. This value is not a handle; that is, the `formPP` value may change if the object moves in memory. In subsequent calls, always use the new `formPP` value returned by this function.

→ *ID*
: Symbolic ID of the control.

→ *style*
: A `ControlStyleType` value specifying the kind of control to create: button, push button, repeating button, check box, popup trigger, or popup selector. To create a graphical control or slider control dynamically, use `CtlNewGraphicControl()` or `CtlNewSliderControl()`, respectively.

`textP`
: Pointer to the control's label text. If `textP` is `NULL`, the control has no label. Only buttons, push buttons, and text boxes have text labels. Because the contents of this pointer are copied into their own buffer, you can free the `textP` pointer any time after the `CtlNewControl` function returns. The buffer into which this string is copied is freed automatically when you remove the control from the form or delete the form.

→ *x*
: Horizontal coordinate of the upper-left corner of the control's boundaries, relative to the window in which it appears.

→ *y*
Vertical coordinate of the upper-left corner of the control's boundaries, relative to the window in which it appears.

→ *width*
Width of the control, expressed in pixels. Valid values are 1–160. If the value of either of the `width` or `height` parameters is 0, the control is sized automatically as necessary to display the text passed as the value of the text parameter.

→ *height*
Height of the control, expressed in pixels. Valid values are 1–160. If the value of either of the `width` or `height` parameters is 0, the control is sized automatically as necessary to display the text passed as the value of the text parameter.

→ *font*
Font used to draw the control's label.

→ *group*
Group ID of a push button or a check box that is part of an exclusive group. The control routines don't turn one control off automatically when another is selected. It's up to the application or a higher-level object, such as a dialog box, to manage this.

→ *leftAnchor*
`true` specifies that the left bound of this control is fixed. This attribute is used by controls that resize dynamically in response to label text changes.

Returns Returns a pointer to the new control.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `CtlValidatePointer()`, `FrmRemoveObject()`

CtlNewGraphicControl Function

Purpose Create a new graphical control dynamically and install it in the specified form.

Declared In `Control.h`

Prototype

```
GraphicControlType *CtlNewGraphicControl
    (void **formPP, UInt16 ID,
    ControlStyleType style,
    DmResID bitmapID, DmResID selectedBitmapID,
    Coord x, Coord y, Coord width, Coord height,
    UInt8 group, Boolean leftAnchor)
```

Parameters

↔ *formPP*
Pointer to the pointer to the form in which the new control is installed. This value is not a handle; that is, the `formPP` value may change if the object moves in memory. In subsequent calls, always use the new `formPP` value returned by this function.

→ *ID*
Symbolic ID of the control.

→ *style*
A `ControlStyleType` value specifying the kind of control to create: button, push button, popup trigger, repeating button, or popup selector. Graphic controls cannot be check boxes.

→ *bitmapID*
Resource ID of the bitmap to display on the control.

→ *selectedBitmapID*
Resource ID of the bitmap to display when the control is selected, if different from `bitmapID`.

→ *x*
Horizontal coordinate of the upper-left corner of the control's boundaries, relative to the window in which it appears.

→ *y*
Vertical coordinate of the upper-left corner of the control's boundaries, relative to the window in which it appears.

→ *width*
Width of the control, expressed in pixels. Valid values are 1–160.

→ *height*
Height of the control, expressed in pixels. Valid values are 1–160.

→ *group*
Group ID of a push button that is part of an exclusive group. The control routines don't turn one control off automatically when another is selected. It's up to the application or a higher-level object, such as a dialog box, to manage this.

→ *leftAnchor*
`true` specifies that the left bound of this control is fixed.

Returns Returns a pointer to the new graphical control. See `GraphicControlType`.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `CtlNewSliderControl()`, `CtlNewControl()`, `CtlValidatePointer()`, `FrmRemoveObject()`

CtlNewSliderControl Function

Purpose Create a new slider or feedback slider dynamically and install it in the specified form.

Declared In `Control.h`

Prototype

```
SliderControlType *CtlNewSliderControl
    void **formPP, UInt16 ID,
    ControlStyleType style,
    DmResID thumbID, DmResID backgroundID,
    Coord x,
    Coord y, Coord width, Coord height,
    UInt16 minValue, UInt16 maxValue,
    UInt16 pageSize,
    UInt16 value)
```

Parameters ↔ *formPP*
Pointer to the pointer to the form in which the new control is installed. This value is not a handle; that is, the `formPP` value may change if the object moves in memory. In subsequent calls, always use the new `formPP` value returned by this function.

→ *ID*
Symbolic ID of the slider.

→ *style*
Either `sliderCtl` or `feedbackSliderCtl`. See `ControlStyleType`.

→ *thumbID*
Resource ID of the bitmap to display as the slider thumb. The slider thumb is the knob that the user can drag to change the slider's value. To use the default thumb bitmap, pass `NULL` for this parameter.

→ *backgroundID*
Resource ID of the bitmap to display as the slider background. To use the default background bitmap, pass `NULL` for this parameter.

→ *x*
Horizontal coordinate of the upper-left corner of the slider's boundaries, relative to the window in which it appears.

→ *y*
Vertical coordinate of the upper-left corner of the slider's boundaries, relative to the window in which it appears.

→ *width*
Width of the slider, expressed in pixels. Valid values are 1–160.

→ *height*
Height of the slider, expressed in pixels. Valid values are 1–160.

→ *minValue*
Value of the slider when its thumb is all the way to the left.

→ *maxValue*
Value of the slider when its thumb is all the way to the right.

→ *pageSize*
Amount by which to increase or decrease the slider's value when the user clicks to the right or left of the thumb.

→ *value*
The initial value to display in the slider.

Returns Returns a pointer to the new slider control. See `SliderControlType`.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `CtlNewGraphicControl()`, `CtlNewControl()`, `CtlValidatePointer()`, `FrmRemoveObject()`

CtlSetEnabled Function

Purpose Set a control as enabled or disabled. Disabled controls do not respond to the pen.

Declared In `Control.h`

Prototype

```
void CtlSetEnabled (ControlType *controlP,
    Boolean usable)
```

Parameters

→ *controlP*
Pointer to a control object. (See `ControlType`.)

→ *usable*
`true` to enable the control; `false` to disable the control.

Returns Returns nothing.

Comments If you disable a visible control, the control is still displayed, and its appearance is no different from controls that do respond to the pen. You might use such a control to inform your users of some state of your application that cannot be modified.

See Also `CtlEnabled()`

CtlSetGraphics Function

Purpose Set the bitmaps for a graphical control and redraw the control if it is visible.

Declared In `Control.h`

Prototype

```
void CtlSetGraphics (ControlType *ctlP,
    DmResID newBitmapID,
    DmResID newSelectedBitmapID)
```

Parameters

→ *ctlP*
Pointer to a graphical control object. (See `GraphicControlType`.)

→ *newBitmapID*
Resource ID of a new bitmap to display on the control, or `NULL` to use the current bitmap.

→ *newSelectedBitmapID*
Resource ID of a new bitmap to display when the control is selected, or `NULL` to use the current selected bitmap.

Returns Returns nothing.

Comments If `ctlP` is not a graphical control, this function immediately returns.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `GraphicControlType`

CtlSetLabel Function

Purpose Set the current label for the specified control object and redraw the control if it is visible.

Declared In `Control.h`

Prototype

```
void CtlSetLabel (ControlType *controlP,
    const Char *newLabel)
```

Parameters

→ *controlP*
Pointer to a control object. (See `ControlType`.)

→ *newLabel*
Pointer to the new text label. Must be a null-terminated string.

Returns Returns nothing.

Comments This function resizes the width of the control to the size of the new label.

This function stores the `newLabel` pointer in the control's data structure. It doesn't make a copy of the string that is passed in. Therefore, if you use `CtlSetLabel`, you must manage the string yourself. You must ensure that it persists for as long as it is being displayed (that is, for as long as the control is displayed or until you call `CtlSetLabel` with a new string), and you must free the string after it is no longer in use (typically after the form containing the control is freed).

If you never use `CtlSetLabel`, you do not need to worry about freeing a control's label.

Make sure that `controlP` is not a graphical control or a slider control. The graphical controls and slider control structures do not contain a text label field, so attempting to set one will crash your application.

See Also CtlGetLabel()

CtlSetSliderValues Function

Purpose Change a slider control's values and redraw the slider if it is visible.

Declared In `Control.h`

Prototype

```
void CtlSetSliderValues (ControlType *ctlP,
    const UInt16 *minValueP,
    const UInt16 *maxValueP,
    const UInt16 *pageSizeP, const UInt16 *valueP)
```

Parameters

→ *`ctlP`*
Pointer to an inactive slider or feedback slider control. (See `SliderControlType`.)

→ *`minValueP`*
Pointer to a new value to use for the slider's minimum or `NULL` if you don't want to change this value.

→ *`maxValueP`*
Pointer to a new value to use for the slider's maximum, or `NULL` if you don't want to change this value.

→ *`pageSizeP`*
Pointer to a new value to use for the slider's page size, or `NULL` if you don't want to change this value.

→ *`valueP`*
Pointer to a new value to use for the current value, or `NULL` if you don't want to change this value.

Returns Returns nothing.

Comments The control's style must be `sliderCtl` or `feedbackSliderCtl`, and it not be currently tracking the pen. If the slider is currently tracking the pen, use `CtlSetValue()` to set the `value` field.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `CtlGetSliderValues()`, `SliderControlType`

CtlSetUsable Function

Purpose Set a control to usable or not usable.

Declared In `Control.h`

Prototype

```
void CtlSetUsable (ControlType *controlP,
    Boolean usable)
```

Parameters

→ *controlP*
Pointer to a control object. (See `ControlType`.)

→ *usable*
`true` to have the control be usable; `false` to have the control be not usable.

Returns Returns nothing.

Comments A control that is not usable doesn't draw and doesn't respond to the pen.

This function doesn't usually update the control.

See Also `CtlEraseControl()`, `CtlHideControl()`, `CtlShowControl()`

CtlSetValue Function

Purpose Set the current value of the specified control. If the control is visible, it's redrawn.

Declared In `Control.h`

Prototype

```
void CtlSetValue (ControlType *controlP,
    Int16 newValue)
```

Parameters

→ *controlP*
Pointer to a control object. (See `ControlType`.)

→ *newValue*
New value to set for the control. For sliders, specify a value between the slider's minimum and maximum. For graphical

controls, push buttons, or check boxes, specify 0 for off, nonzero for on.

Returns Returns nothing.

Comments This function works only with graphical controls, sliders, push buttons, and check boxes. If you set the value of any other type of control, the behavior is undefined.

Compatibility Sliders and graphical controls are only supported if 3.5 New Feature Set is present.

See Also `CtlGetValue()`, `FrmGetControlGroupSelection()`, `FrmSetControlGroupSelection()`, `FrmGetControlValue()`, `FrmSetControlValue()`

CtlShowControl Function

Purpose Set a control's `usable` attribute to `true` and draw the control on the screen. This function calls `CtlDrawControl()`.

Declared In `Control.h`

Prototype

```
void CtlShowControl (ControlType *controlP)
```

Parameters → *controlP*
Pointer to a control object. (See `ControlType`.)

Returns Returns nothing.

Comments If the control is already usable, this function is the functional equivalent of `CtlDrawControl`.

Sets the `visible` and the `usable` attributes to `true`. (See `ControlAttrType`.)

Don't use this function directly; instead use `FrmShowObject()`, which does the same thing.

See Also `CtlHideControl()`

CtlValidatePointer Function

Purpose Returns `true` if the specified pointer references a valid control object.

Declared In `Control.h`

Prototype

```
Boolean CtlValidatePointer
    (const ControlType *controlP)
```

Parameters → *`controlP`*
Pointer to a control. (See `ControlType`.)

Returns Returns `true` when passed a valid pointer to a control; otherwise, returns `false`.

Comments For debugging purposes; do not include this function in commercial products. In debug builds, this function displays a dialog and waits for the debugger when an error occurs.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also FrmValidatePtr(), WinValidateHandle()

8

Date and Time Selector

The Palm OS® UI provides two system resources for accepting date and time input values. These resources are dialog boxes that contain UI gadgetry for entering dates and times. The Palm OS UI also provides functions to manage the interaction with these resources. This chapter describes those functions.

The API described in this chapter is declared in the header files `Day.h`, `SelDay.h`, `SelTime.h`, and `SelTimeZone.h`.

Date and Time Selections Data Structures

SelectDayType Struct

Prototype

```
typedef enum {
  selectDayByDay,     // return d/m/y
  selectDayByWeek,  // return d/m/y with d as
                     // same day of the week
  selectDayByMonth // return d/m/y with d as
                    // same day of the month
} SelectDayType;
```

DaySelectorType Struct

Prototype

```
typedef struct DaySelectorType {
  RectangleType  bounds;
  Boolean        visible;
  UInt8          reserved1;
  Int16         visibleMonth; // month actually
                                // displayed
  Int16          visibleYear;  // year actually
```

```
                                        // displayed
  DateTimeType   selected;
  SelectDayType  selectDayBy;
  UInt8          reserved2;
} DaySelectorType;
```

HMSTime Struct

Prototype

```
typedef struct {
  UInt8 hours;
  UInt8 minutes;
  UInt8 seconds;
  UInt8 reserved;
} HMSTime;
```

Date and Time Selection Functions

DayDrawDays Function

Purpose Draw only the days-of-the-month portion of a day selector control object.

Declared In `Day.h`

Prototype

```
void DayDrawDays
    (const DaySelectorType *selectorP)
```

Parameters

→ *selectorP*
Pointer to the control object to draw.

Returns Nothing.

Comments This function is used when the year or month changes. Only drawing the portion of the control that presents the days of the month avoids the flicker that would occur if the week titles were redrawn.

Compatibility If either 5.0 New Feature Set or Palm OS Cobalt Feature Set is present this function is unimplemented.

See Also `DayDrawDaySelector()`

DayDrawDaySelector Function

Purpose Draw a day selector control object on screen.

Declared In `Day.h`

Prototype

```
void DayDrawDaySelector
    (const DaySelectorType *selectorP)
```

Parameters → *selectorP*
Pointer to the control object to draw.

Returns Nothing.

Comments The control is drawn only if it is usable.

Compatibility If either 5.0 New Feature Set or Palm OS Cobalt Feature Set is present this function is unimplemented.

See Also `DayDrawDays()`

DayHandleEvent Function

Purpose Handle event in the specified control. This function handles two types of events, `penDownEvent` and `ctlEnterEvent`. It is primarily intended for use by the operating system; applications do not typically use this function.

Declared In `Day.h`

Prototype

```
Boolean DayHandleEvent
    (DaySelectorType *selectorP,
    const EventType *pEvent)
```

Parameters → *selectorP*
Pointer to control object.

→ *pEvent*
Pointer to an `EventType` structure.

Returns `true` if the event was handled or `false` if it was not.

Posts a `daySelectEvent` with information on whether to use the date.

Comments A date is used if the user selects a day in the visible month.

When this function receives a `penDownEvent`, it checks if the pen position is within the bounds of the control object. If it is, a

`dayEnterEvent` is added to the event queue and the function exits.

When this function receives a `dayEnterEvent`, it checks that the control id in the event record matches the id of the control specified. If they match, this function will track the pen until it comes up in the bounds in which case daySelectEvent is sent.

If the pen exits the bounds a `dayExitEvent` is sent.

Compatibility If Palm OS Cobalt Feature Set is present this function is unimplemented.

SelectDay Function

Purpose Display a form showing a date; allow user to select a different date.

Declared In `SelDay.h`

Prototype

```
Boolean SelectDay
    (const SelectDayType selectDayBy,
    Int16 *month,
    Int16 *day, Int16 *year, const Char *title)
```

Parameters

`selectDayBy`
The method by which the user should choose the day. Possible values are `selectDayByDay`, `selectDayByWeek`, and `selectDayByMonth`. See `SelectDayType`

↔ *`month, day, year`*
Month, day, and year selected.

→ *`title`*
String title for the dialog.

Returns `true` if the OK button was pressed. If `true`, `month`, `day`, and `year` contain the new date.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `SelectDayV10()`

SelectDayV10 Function

Purpose Display a form showing a date, allow user to select a different date.

Declared In `SelDay.h`

Prototype

```
Boolean SelectDayV10 (Int16 *month, Int16 *day,
   Int16 *year, const Char *title)
```

Parameters

↔ *month, day, year*
Month, day, and year selected. The initial values passed in these parameters must be valid.

→ *title*
String title for the dialog.

Returns Returns `true` if the OK button was pressed. In that case, the parameters passed are changed.

Compatibility This function corresponds to the 1.0 version of `SelectDay`.

See Also `SelectDay()`

SelectOneTime Function

Purpose Display a form showing the time and allow the user to select a different time.

Declared In `SelTime.h`

Prototype

```
Boolean SelectOneTime (Int16 *hour,
   Int16 *minute, const Char *titleP)
```

Parameters

↔ *hour*
The hour selected in the form.

↔ *minute*
The minute selected in the form.

→ *titleP*
A pointer to a string to display as the title. Doesn't change as the function executes.

Returns Returns `true` if the user selects OK and `false` otherwise. If `true` is returned, the values in `hour` and `minute` have probably been changed.

Comments Use this function instead of `SelectTime` if you want to display a dialog that specifies a single point in time, not a range of time from start to end.

Compatibility Implemented only if 3.1 New Feature Set is present.

See Also `SelectTimeV33()`

SelectTime Function

Purpose Display a form showing a start and end time. Allow the user to select a different time.

Declared In `SelTime.h`

Prototype

```
Boolean SelectTime (TimeType *startTimeP,
   TimeType *EndTimeP, Boolean untimed, const
   Char
   *titleP, Int16 startOfDay, Int16 endOfDay,
   Int16 startOfDisplay)
```

Parameters

↔ *startTimeP, EndTimeP*
: Pointers to values of type `TimeType`. Pass values to display in these two parameters. If the user makes a selection and taps the OK button, the selected values are returned here.

→ *untimed*
: Pass in `true` to indicate that no time is selected. If the user sets the time to no time then `startTimeP` and `EndTimeP` are both set to the constant `noTime (-1)`.

→ *titleP*
: A pointer to a string to display as the title. Doesn't change as the function executes.

→ *startOfDay*
: The hour that the hour list displays at its top. To see earlier hours, the user can scroll the list up. The value must be between 0 to 12, inclusive.

→ *endOfDay*
: `The` hour used when the "All Day" button is selected.

→ *startOfDisplay*
: First hour initially visible.

Returns Returns `true` if the user selects OK and `false` otherwise. If `true` is returned, the values in `hour` and `minute` have probably been changed.

Comments This version of `SelectTime` adds the `endOfDay` and `startOfDisplay` functionality.

Compatibility Implemented if 3.5 New Feature Set is present.

See Also SelectDay(), SelectOneTime()

SelectTimeV33 Function

Purpose Display a form showing the time and allow the user to select a different time.

This function is obsolete and should not be used.

Declared In `SelTime.h`

Prototype

```
Boolean SelectTimeV33 (TimeType *startTimeP,
   TimeType *EndTimeP, Boolean untimed,
   const Char *titleP, Int16 startOfDay)
```

Parameters

↔ *`startTimeP, EndTimeP`*
Pointers to values of type `TimeType`. Pass values to display in these two parameters. If the user makes a selection and taps the OK button, the selected values are returned here.

→ *`untimed`*
Pass in `true` to indicate that no time is selected. If the user sets the time to no time then `startTimeP` and `EndTimeP` are both set to the constant `noTime (-1)`.

→ *`titleP`*
A pointer to a string to display as the title. Doesn't change as the function executes.

→ *`startOfDay`*
The hour that the hour list displays at its top. To see earlier hours, the user can scroll the list up. The value must be between 0 to 12, inclusive.

Returns Returns `true` if the user selects OK and `false` otherwise. If `true` is returned, the values in `hour` and `minute` have probably been changed.

Comments

NOTE: Obsolete functions are provided ONLY for backward compatibility; for example, so a 1.0 application will work on 3.x OS releases. New code should not call these functions!

See Also SelectDay(), SelectOneTime()

SelectTimeZone Function

Purpose Display a form that allows the user to select a different time zone.

Declared In `SelTimeZone.h`

Prototype

```
Boolean SelectTimeZone (Int16 *ioTimeZoneP,
   LmLocaleType *ioLocaleInTimeZoneP,
   const Char *titleP, Boolean showTimes,
   Boolean anyLocale)
```

Parameters

↔ *ioTimeZoneP*
: A pointer to the time zone, given as minutes east of Greenwich Mean Time (GMT). The initial value is used as the initial selection in the form. Upon return, this parameter contains a pointer to the new time zone that the user selected.

↔ *ioLocaleInTimeZoneP*
: A pointer to a locale (see `LmLocaleType`) that identifies the time zone country. This parameter is used for countries that share a time zone, such as Canada and Chile.

  If the time zone specified by `ioTimeZoneP` is specific to a particular country, you do not have to initialize this parameter. Instead, set the `anyLocale` parameter to `true` to have this parameter ignored upon entry.

→ *titleP*
: A string to use as the title for the dialog. Pass `NULL` to use the default title, which is "Set Time Zone".

→ *showTimes*
: If `true`, the dialog shows the correct times in both the current and newly selected time zones. If `false`, the dialog doesn't show the current time. Using `false` provides a larger area for the list of time zones.

→ *anyLocale*
: If `true`, ignore `ioLocaleInTimeZoneP` upon entry.

Returns Returns `true` if the user clicked the OK button in the dialog to change the time zone, or `false` if the user clicked the Cancel button.

Comments The time zones displayed in the form are listed by country. For this reason, if the time zone specified by `ioTimeZoneP` is shared by several countries, you need to supply a value for `ioLocaleTimeZoneP` to identify which country should be selected when the list is first displayed. You can use the constant `lmAnyLanguage` as the value for the language field of the structure pointed to by this parameter.

If you don't care which value is initially selected, pass `true` for the `anyLocale` parameter. In this case, the first country that matches the GMT offset given in `ioTimeZoneP` is selected.

You might want to use the current time zone stored in the system preferences as the initial value for the `ioTimeZoneP` parameter. To obtain this time zone, do the following:

```
Int16 timeZone =
  (Int16)PrefGetPreference(prefTimeZone);
CountryType timeZoneCountry = (CountryType)
  PrefGetPreference(prefTimeZoneCountry);
LmLocaleType timeZoneLocale;
Boolean change;

timeZoneLocale.country = timeZoneCountry;
timeZoneLocale.language = lmAnyLanguage;
change = SelectTimeZone(&timeZone,
  &timeZoneLocale, NULL, true, false);
```

Compatibility Implemented if 4.0 New Feature Set is present.

9

Fields

This chapter provides the following information about field objects:

- Field Data Structures
- Field Resources
- Field Functions

The header file `Field.h` declares the API that this chapter describes. For more information on fields, see the section "Fields" in the *Palm OS Programmer's Companion*, vol. I.

Field Data Structures

FieldAttrType Struct

Purpose The `FieldAttrType` bit field defines the visible characteristics of the field. The functions `FldGetAttributes()` and `FldSetAttributes()` return and set these values. There are other functions that retrieve or set individual attributes defined here. Those functions are noted below.

WARNING! You must not call `FldSetAttributes()` to change any attributes that are noted as "for internal use only."

Prototype

```
typedef struct {
  UInt16 usable:1;
  UInt16 visible:1;
  UInt16 editable:1;
  UInt16 singleLine:1;
  UInt16 hasFocus:1;
  UInt16 dynamicSize:1;
  UInt16 insPtVisible:1;
  UInt16 dirty:1;
  UInt16 underlined:2;
```

```
  UInt16 justification:2;
  UInt16 autoShift:1;
  UInt16 hasScrollBar:1;
  UInt16 numeric:1;
} FieldAttrType;
```

Fields

`usable`
: If not set, the field object is not considered part of the current interface of the application, and it doesn't appear on screen. The function `FldSetUsable()` sets this value, but it is better to use `FrmShowObject()`.

`visible`
: Set or cleared internally when the field object is drawn with `FldDrawField()` or `FrmShowObject()`, and erased with `FldEraseField()` or `FrmHideObject()`.

`editable`
: If not set, the field object doesn't accept text input or editing commands and the insertion point cannot be positioned with the pen. The text can still be selected and copied.

`singleLine`
: (This flag is for internal use only) If set, the field is a single line of text high and the text does not wrap when it exceeds the width of the field. If not set, the text wraps to fill multiple lines. You cannot change the field type from single-line to multi-line, or vice versa, by changing the value of this flag.

`hasFocus`
: Set internally when the field has the current focus. The blinking insertion point appears in the field that has the current focus. Use the function `FrmSetFocus()` and `FldReleaseFocus()` to set this value. Do *not* try to set this attribute with `FldSetAttributes`.

`dynamicSize`
: If set, a `fldHeightChangedEvent` is generated whenever the number of lines needs to increase or decrease. Your application needs to respond to this event by adjusting the size of the field's bounding box. If not set, the text wraps to fill more (or less) lines as required, but the event is not generated. *Note that this bit does not cause the field to change size automatically*; your application must respond to the `fldHeightChangedEvent` and resize the field itself.

  Set this attribute to `false` if the Single Line attribute is set.

`insPtVisible`
: If set, the insertion point is scrolled into view. This attribute is for internal use only..

`dirty`
: If set, the user has modified the field. The functions `FldDirty()` and `FldSetDirty()` retrieve this field's value. Do *not* try to set this attribute with `FldSetAttributes`.

`underlined`
: If set each line of the field, including blank lines, is underlined. Possible values are defined by the `UnderlineModeType` defined in `Window.h`:

  `noUnderline`

  `grayUnderline`

  `solidUnderline`

  `colorUnderline`

  Editable text fields generally use `grayUnderline` as the value.

  The `solidUnderline` value is only valid for Palm OS 3.1 and higher.

  The `colorUnderline` value is only valid for Palm OS 3.5 and higher.

`justification`
: Specifies the text alignment. Possible values are `leftAlign` and `rightAlign`. (left or right justification only; `centerAlign` justification is not supported).

`autoShift`
: If set, auto-shift rules are applied.

`hasScrollBar`
: If set, the field has a scrollbar. The system sends more frequent `fldChangedEvent`s so the application can adjust the height appropriately.

`numeric`
: If set, only the characters 0 through 9 and associated separators are allowed to be entered in the field. The associated separators are the thousands separator and the

decimal character. The values of these two characters depend on the settings in the Formats prefs panel.

FieldPtr Typedef

Purpose The `FieldPtr` type defines a pointer to a `FieldType` structure.

Prototype

```
typedef FieldType *FieldPtr;
```

You pass the `FieldPtr` as an argument to all field functions. You can obtain the `FieldPtr` using the function `FrmGetObjectPtr()` in this way:

```
fldPtr = FrmGetObjectPtr(frm,
  FrmGetObjectIndex(frm, fldID));
```

where `fldID` is the resource ID assigned when you created the field.

FieldType Struct

Purpose The `FieldType` structure represents a field.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `FieldType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  UInt16 id;
  RectangleType rect;
  FieldAttrType attr;
  Char *text;
  MemHandle textHandle;
  LineInfoPtr lines;
  UInt16 textLen;
  UInt16 textBlockSize;
  UInt16 maxChars;
  UInt16 selFirstPos;
  UInt16 selLastPos;
```

```
  UInt16 insPtXPos;
  UInt16 insPtYPos;
  FontID fontID;
  UInt8 reserved;
} FieldType;
```

Your code should treat the `FieldType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

Fields

`id`

ID value you specified when you created the field resource. This ID value is included as part of the event data of `fldEnterEvent`.

`rect`

Position and size of the field object. The functions `FldGetBounds()`, `FrmGetObjectBounds()`, `FldSetBounds()`, and `FrmSetObjectBounds()` retrieve and set this value.

`attr`

Field object attributes. (See `FieldAttrType`.)

`text`

Pointer to the null-terminated string that is displayed by the field object. The functions `FldGetTextPtr()` and `FldSetTextPtr()` retrieve and set this value (see below).

Never set the contents of this string directly; for example, do not pass this pointer as the destination value to a function such as `StrCopy`.

`textHandle`

Handle to the stored text or to a database record containing the stored text. The functions `FldGetTextHandle()` and `FldSetTextHandle()` retrieve and set this value.

If `textHandle` is defined, the field calculates the `text` pointer when it locks the handle. In general, you should only use `FldGetTextPtr()` and `FldSetTextPtr()` on text fields that aren't editable. On editable text fields, use `FldGetTextHandle()` and `FldSetTextHandle()`.

Also note that editable text fields allocate the text handle as necessary. If a user starts typing in a field that doesn't have a text handle allocated, the field will allocate one. The field also

resizes the text's memory block as necessary when the user adds more text.

`lines`
: Pointer to an array of `LineInfoType` structures. There is one entry in this array for each visible line of the text. (See `LineInfoType`.) The field code maintains this array internally; you should never change the `lines` array yourself.

  Note that this value is `NULL` for single line fields, and for fields that do not have an allocated text handle.

`textLen`
: Length in bytes of the string currently displayed by the field object; the null terminator is excluded. You can retrieve this value with `FldGetTextLength()`.

`textBlockSize`
: Allocated size of the memory block that holds the field object's text string. You can retrieve this value with `FldGetTextAllocatedSize()`.

  Fields allocate memory for the field text as needed, several bytes at a time.

  Note that `textBlockSize` may be different from the size of the chunk pointed to by `textHandle`. The `textHandle` may point to a database record that contains, in part, the text displayed by the field. If you called `MemHandleSize()` on such a `textHandle`, the number returned may be greater than `textBlockSize`.

`maxChars`
: Maximum number of bytes the field object accepts. The functions `FldGetMaxChars()` and `FldSetMaxChars()` retrieve and set this value.

  Note the difference between `textLen`, `textBlockSize`, and `maxChars`. `textLen` is the number of bytes of character data that `text` actually holds. `textBlockSize` is the amount of memory currently allocated for the text (which must be greater than or equal to `textLen`), and `maxChars` sets the maximum value that `textBlockSize` and `textLen` can expand to.

  For example, if you've created a text field for users to enter their first names in, you might specify that the maximum

length of this field is 20 bytes. If a user enters "John" in this field, `textLen` is 4, `textBlockSize` is 16, and `maxChars` is 20.

`selFirstPos`
: Starting character offset in bytes of the current selection. Use `FldGetSelection()` and `FldSetSelection()` to retrieve and set this value and the `selLastPos` value.

`selLastPos`
: Ending character offset in bytes of the current selection. When `selFirstPos` equals `selLastPos`, there is no selection.

`insPtXPos`
: Horizontal location of the insertion point, given as the offset in bytes into the line indicated by `insPtYPos`. The functions `FldGetInsPtPosition()` and `FldSetInsPtPosition()` retrieve and set a byte offset calculated from this value. If the insertion point isn't visible—if it's on a line that's either above or below the set of visible lines—`insPtXPos` is the absolute byte offset of the insertion point.

`insPtYPos`
: Vertical location of the insertion point, given as the display line where the insertion point is positioned. The first display line is zero. The first display line may be different from the first line of text in the field if the field has been scrolled. The functions `FldGetInsPtPosition()` and `FldSetInsPtPosition()` retrieve and set a byte offset calculated from this value. If the insertion point isn't visible—if it's on a line that's either above or below the set of visible lines—`insPtYPos` is set to `0x8000`.

`fontID`
: Font ID for the field. See `Font.h` for more information. The functions `FldGetFont()` and `FldSetFont()` retrieve and set this value.

`reserved`
: Reserved for future use.

LineInfoPtr Typedef

Purpose The `LineInfoPtr` type defines a pointer to the `LineInfoType`.

Prototype

```
typedef LineInfoType *LineInfoPtr;
```

LineInfoType Struct

Purpose The `LineInfoType` structure defines an element in the field's `lines` array. The `lines` array contains the field's word wrapping information. There is one element in the array per visible line in the field, including visible lines that contain no text. The field code maintains this array internally; you should never change the `lines` array yourself.

The functions `FldCalcFieldHeight()`, `FldGetVisibleLines()`, `FldRecalculateField()`, and `FldGetNumberOfBlankLines()` retrieve or set information in this structure. The scrolling functions `FldGetScrollPosition()`, `FldGetScrollValues()`, `FldScrollField()`, and `FldSetScrollPosition()` also retrieve or set information in this structure.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `LineInfoType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  UInt16 start;
  UInt16 length;
} LineInfoType;
```

Fields

`start`
: The byte offset into the `FieldType`'s `text` field of the first character displayed by this line. If the line is blank, `start` is equal to `textLen` and `length` is 0.

`length`
: The length in bytes of the portion of the string displayed on this line. If the line is blank, the length is 0.

Field Resources

The Field Resource (tFLD) represents a field on screen. See *Palm OS User Interface Guidelines* for more information.

Field Functions

FldCalcFieldHeight Function

Purpose Determine the height of a field for a string.

Declared In `Field.h`

Prototype

```
UInt16 FldCalcFieldHeight (const Char *chars,
    UInt16 maxWidth)
```

Parameters

→ *chars*
Pointer to a null-terminated string.

→ *maxWidth*
Maximum line width in pixels.

Returns Returns the total number of lines needed to draw the string passed.

Comments The width of a field is contained in the `rect` member of the `FieldType` structure. You can retrieve this value in the following way:

```
FrmGetObjectBounds(frm,
  FrmGetObjectIndex(frm, fldID),
  &myRect);
fieldWidth = myRect.extent.x;
FldCalcFieldHeight(myString, fieldWidth);
```

See Also `FldWordWrap()`

FldCompactText Function

Purpose Compact the memory block that contains the field's text to release any unused space.

Declared In `Field.h`

Prototype
```
void FldCompactText (FieldType *fldP)
```

Parameters → *fldP*
Pointer to a field object (`FieldType` structure).

Returns Returns nothing.

Comments As characters are added to the field's text, the block that contains the text is grown. The block is expanded several bytes at a time so that it doesn't have to expand each time a character is added. This expansion may result in some unused space in the text block.

Applications should call this function on field objects that edit data records in place before the field is unlocked, or at any other time when a compact field is desirable; for example, before writing to the storage heap.

See Also `FldGetTextAllocatedSize()`, `FldSetTextAllocatedSize()`

FldCopy Function

Purpose Copy the current selection to the text clipboard.

Declared In `Field.h`

Prototype
```
void FldCopy (const FieldType *fldP)
```

Parameters → *fldP*
Pointer to a field object (`FieldType` structure).

Returns Returns nothing.

Comments This function leaves the current selection highlighted.

This function replaces anything previously in the text clipboard if there is text to copy. If no text is selected, the function beeps and the clipboard remains intact.

See Also `FldCut()`, `FldPaste()`

FldCut Function

Purpose Copy the current selection to the text clipboard, delete the selection from the field, and redraw the field.

Declared In `Field.h`

Prototype

```
void FldCut (FieldType *fldP)
```

Parameters

→ *fldP*
Pointer to a field object (`FieldType` structure).

Returns Returns nothing.

Comments If text is selected, the text is removed from the field, the field's dirty attribute is set, and anything previously in the text clipboard is replaced by the selected text.

If there is no selection or if the field is not editable, this function beeps.

See Also `FldCopy()`, `FldPaste()`, `FldUndo()`

FldDelete Function

Purpose Delete the specified range of characters from the field and redraw the field.

Declared In `Field.h`

Prototype

```
void FldDelete (FieldType *fldP, UInt16 start,
    UInt16 end)
```

Parameters

→ *fldP*
Pointer to the field object (`FieldType` structure) to delete from.

→ *start*
The beginning of the range of characters to delete given as a valid byte offset into the field's `text` string.

→ *end*
The end of the range of characters to delete given as a valid byte offset into the field's `text` string. On systems that support multi-byte characters, this position must be an inter-

character boundary. That is, it must not point to a middle byte of a multi-byte character.

If you pass a value that is greater than the number of bytes in the field, all characters in the field are deleted.

Returns Returns nothing.

Comments This function deletes all characters from the starting offset up to the ending offset and sets the field's dirty attribute. It does not delete the character at the ending offset.

If `start` or `end` point to an intra-character boundary, `FldDelete` attempts to move the offset backward, toward the beginning of the text, until the offset points to an inter-character boundary (i.e., the start of a character).

`FldDelete` posts a `fldChangedEvent` to the event queue. If you call this function repeatedly, you may overflow the event queue with `fldChangedEvents`. An alternative is to remove the text handle from the field, change the text, and then set the field's handle again. See `FldGetTextHandle()` for a code example.

See Also `FldInsert()`, `FldEraseField()`, `TxtCharBounds()`

FldDirty Function

Purpose Return `true` if the field has been modified since the text value was set.

Declared In `Field.h`

Prototype

```
Boolean FldDirty (const FieldType *fldP)
```

Parameters → *fldP*

Pointer to a field object (`FieldType` structure).

Returns Returns `true` if the field has been modified either by the user or through calls to certain functions such as `FldInsert()` and `FldDelete()`, `false` if the field has not been modified.

See Also `FldSetDirty()`, `FieldAttrType`

FldDrawField Function

Purpose Draw the text of the field.

Declared In `Field.h`

Prototype
```
void FldDrawField (FieldType *fldP)
```

Parameters → *fldP*
Pointer to a field object (`FieldType` structure).

Returns Returns nothing.

Comments The field's `usable` attribute must be `true` or the field won't be drawn.

This function doesn't erase the area behind the field before drawing.

If the field has the focus, the blinking insertion point is displayed in the field.

See Also `FldEraseField()`

FldEraseField Function

Purpose Erase the text of a field and turn off the insertion point if it's in the field.

Declared In `Field.h`

Prototype
```
void FldEraseField (FieldType *fldP)
```

Parameters → *fldP*
Pointer to a field object (`FieldType` structure).

Returns Returns nothing.

Comments You rarely need to call this function directly. Instead, use `FrmHideObject()`, which calls `FldEraseField` for you.

This function visibly erases the field from the display, but it doesn't modify the contents of the field or free the memory associated with it.

If the field has the focus, the blinking insertion point is turned off.

This function sets the `visible` attribute to `false`. (See `FieldAttrType`.)

See Also `FldDrawField()`

FldFreeMemory Function

Purpose Release the handle-based memory allocated to the field's text and the associated word-wrapping information.

Declared In `Field.h`

Prototype

```
void FldFreeMemory (FieldType *fldP)
```

Parameters → *fldP*
Pointer to a field object (`FieldType` structure).

Returns Returns nothing. May raise a fatal error message if the text associated with the field is actually a record in a database.

Comments This function releases

- The memory allocated to the text of a field—the memory block that the `textHandle` member of the `FieldType` data structure points to.

  If the field's `textHandle` is `NULL` but there is a `text` string associated with that field (which is often the case with noneditable text fields), the text string is not freed.
- The memory allocated to hold the word-wrapping information—the memory block that the `lines` member of the `FieldType` data structure points to.

This function doesn't affect the display of the field. Fields allocate memory for the text string as needed, so it is not an error to call this function while the field is still displayed. That is, if `text` is `NULL` and the user starts typing in the field, the field simply allocates memory for text and continues.

FldGetAttributes Function

Purpose Return the attributes of a field.

Declared In `Field.h`

Prototype

```
void FldGetAttributes (const FieldType *fldP,
    FieldAttrPtr attrP)
```

Parameters → *fldP*
Pointer to a `FieldType` structure.

← *attrP*
Pointer to the `FieldAttrType` structure.

Returns Returns the field's attributes in the `attrP` parameter.

See Also `FldSetAttributes()`

FldGetBounds Function

Purpose Return the current bounds of a field.

Declared In `Field.h`

Prototype

```
void FldGetBounds (const FieldType *fldP,
    RectanglePtr rect)
```

Parameters → *fldP*
Pointer to a field object (`FieldType` structure).

← *rect*
Pointer to a `RectangleType` structure.

Returns Returns nothing. Stores the field's bounds in the `RectangleType` structure reference by `rect`.

Comments Returns the `rect` field of the `FieldType` structure.

See Also `FldSetBounds()`, `FrmGetObjectBounds()`

FldGetFont Function

Purpose Return the ID of the font used to draw the text of a field.

Declared In `Field.h`

Prototype

```
FontID FldGetFont (const FieldType *fldP)
```

Parameters → *fldP*
Pointer to a field object (`FieldType` structure).

Returns Returns the ID of the font.

See Also `FldSetFont()`

FldGetInsPtPosition Function

Purpose Return the insertion point position within the string.

Declared In `Field.h`

Prototype

```
UInt16 FldGetInsPtPosition
    (const FieldType *fldP)
```

Parameters → *fldP*
Pointer to a field object (`FieldType` structure).

Returns Returns the byte offset of the insertion point.

Comments The insertion point is to the left of the byte offset that this function returns. That is, if this function returns 0, the insertion point is to the left of the first character in the string. In multiline fields, line feeds are counted as a single character in the string, and the byte offset after the line feed character is the beginning of the next line.

See Also `FldSetInsPtPosition()`

FldGetMaxChars Function

Purpose Return the maximum number of bytes the field accepts.

Declared In `Field.h`

Prototype

```
UInt16 FldGetMaxChars (const FieldType *fldP)
```

Parameters → *fldP*
Pointer to a field object (`FieldType` structure).

Returns Returns the maximum length in bytes of characters the user is allowed to enter. This is the `maxChars` field in `FieldType`.

See Also `FldSetMaxChars()`

FldGetNumberOfBlankLines Function

Purpose Return the number of blank lines that are displayed at the bottom of a field.

Declared In `Field.h`

Prototype

```
UInt16 FldGetNumberOfBlankLines
    (const FieldType *fldP)
```

Parameters → *fldP*
Pointer to a `FieldType` structure.

Returns Returns the number of blank lines visible.

Comments This routine is useful for updating a scroll bar after characters have been removed from the text in a field. See the `NoteViewScroll` function in the Address sample application for an example.

Compatibility Implemented only if 2.0 New Feature Set is present.

FldGetScrollPosition Function

Purpose Return the offset of the first character in the first visible line of a field.

Declared In `Field.h`

Prototype

```
UInt16 FldGetScrollPosition
    (const FieldType *fldP)
```

Parameters → *fldP*
Pointer to a field object (`FieldType` structure).

Returns Returns the offset of the first visible character.

See Also `FldSetScrollPosition()`, `LineInfoType`

FldGetScrollValues Function

Purpose Return the values necessary to update a scroll bar.

Declared In `Field.h`

Prototype

```
void FldGetScrollValues (const FieldType *fldP,
    UInt16 *scrollPosP, UInt16 *textHeightP,
    UInt16 *fieldHeightP)
```

Parameters

→ *fldP*
Pointer to a `FieldType` structure.

← *scrollPosP*
The line of text that is the topmost visible line. Line numbering starts with 0.

← *textHeightP*
The number of lines needed to display the field's text, given the width of the field.

← *fieldHeightP*
The number of visible lines in the field.

Returns Returns nothing. Stores the position, text height, and field height in the parameters passed in.

Comments Use the values returned by this function to calculate the values you send to `SclSetScrollBar()` to update the scroll bar. For example:

```
FldGetScrollValues (fldP, &scrollPos,
  &textHeight,  &fieldHeight);

if (textHeight > fieldHeight)
  maxValue = textHeight - fieldHeight;
else if (scrollPos)
  maxValue = scrollPos;
else
  maxValue = 0;

SclSetScrollBar (bar, scrollPos, 0, maxValue,
  fieldHeight-1);
}
```

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `FldSetScrollPosition()`

FldGetSelection Function

Purpose Return the current selection of a field.

Declared In `Field.h`

Prototype

```
void FldGetSelection (const FieldType *fldP,
    UInt16 *startPosition, UInt16 *endPosition)
```

Parameters

→ *fldP*
Pointer to a field object (`FieldType` structure).

← *startPosition*
Pointer to the start of the selected characters range, given as the byte offset into the field's text.

← *endPosition*
Pointer to end of the selected characters range given as the byte offset into the field's text.

Returns Returns the starting and ending byte offsets in `startPosition` and `endPosition`.

Comments The first character in a field is at offset zero.

If the user has selected the first five characters of a field, `startPosition` will contain the value 0 and `endPosition` the value 5, assuming all characters are a single byte long.

See Also `FldSetSelection()`

FldGetTextAllocatedSize Function

Purpose Return the number of bytes allocated to hold the field's text string. Don't confuse this number with the actual length of the text string displayed in the field.

Declared In `Field.h`

Prototype

```
UInt16 FldGetTextAllocatedSize
    (const FieldType *fldP)
```

Parameters

→ *fldP*
Pointer to a field object.

Returns Returns the number of bytes allocated for the field's text. This is the `textBlockSize` field in `FieldType`.

See Also `FldSetTextAllocatedSize()`

FldGetTextHandle Function

Purpose Return a handle to the block that contains the text string of a field.

Declared In `Field.h`

Prototype

```
MemHandle FldGetTextHandle
    (const FieldType *fldP)
```

Parameters → *fldP*

Pointer to a field object (`FieldType` structure).

Returns Returns the handle to the text string of a field or `NULL` if no handle has been allocated for the field pointer.

Comments The handle returned by this function is not necessarily the handle to the start of the string. If you've used `FldSetText()` to set the field's text to a string that is part of a database record, the text handle points to the start of that record. You'll need to compute the offset from the start of the record to the start of the string. You can either store the offset that you passed to `FldSetText` or you can compute the offset by performing pointer arithmetic on the pointer you get by locking this handle and the pointer returned by `FldGetTextPtr()`.

If you are obtaining the text handle so that you can edit the field's text, you must remove the handle from the field before you do so. If you change the text while it is being used by a field, the field's internal structures specifying the text length, allocated size, and word wrapping information can become out of sync. To avoid this problem, remove the text handle from the field, change the text, and then set the field's text handle again. For example:

```
// Get the handle for the string and unlock
// it by removing it from the field.
textH = FldGetTextHandle(fldP);
FldSetTextHandle (fldP, NULL);

/* Insert code that modifies the string here.*/
/* The basic steps are: */
/* resize the chunk if necessary,*/
/* lock the chunk, write to it, and then */
/* unlock the chunk. If the text is in a */
/* database record, use Data Manager calls. */

/* Update the text in the field. */
```

```
FldSetTextHandle (fldP, textH);
FldDrawField(fldP);
```

See Also FldSetTextHandle(), FldGetTextPtr()

FldGetTextHeight Function

Purpose Return the height in pixels of the number of visible lines that are not empty.

Declared In `Field.h`

Prototype

```
UInt16 FldGetTextHeight (const FieldType *fldP)
```

Parameters → *fldP*
Pointer to a field object (FieldType structure).

Returns Returns the height in pixels of the number of visible lines that are not empty.

Comments Empty lines are all of the lines in the field following the last byte of text. Note that lines that contain only a linefeed are not empty. Also note that only lines that are visible are counted.

See Also FldCalcFieldHeight()

FldGetTextLength Function

Purpose Return the length in bytes of the field's text.

Declared In `Field.h`

Prototype

```
UInt16 FldGetTextLength (const FieldType *fldP)
```

Parameters → *fldP*
Pointer to a field object (FieldType structure).

Returns Returns the length in bytes of a field's text, not including the terminating null character. This is the `textLen` field of `FieldType`.

FldGetTextPtr Function

Purpose Return a locked pointer to the field's text string.

Declared In `Field.h`

Prototype
```
Char *FldGetTextPtr (const FieldType *fldP)
```

Parameters → *fldP*
Pointer to a field object (`FieldType` structure).

Returns Returns a locked pointer to the field's text string or `NULL` if the field is empty.

Comments The pointer returned by this function can become invalid if the user edits the text after you obtain the pointer.

Do not modify the contents of the pointer yourself. If you change the text while it is being used by a field, the field's internal structures specifying the text length, allocated size, and word wrapping information can become out of sync. To avoid this problem, follow the instructions given under `FldGetTextHandle()`.

WARNING! The pointer returned by this function is "owned" by the field until you specify a different pointer for the field. You should not store this pointer for future use, since the field can modify the size of the string, which can cause the pointer to become invalid.

See Also `FldSetTextPtr()`, `FldGetTextHandle()`

FldGetVisibleLines Function

Purpose Return the number of lines that can be displayed within the visible bounds of the field, regardless of what text is stored in the field.

Declared In `Field.h`

Prototype
```
UInt16 FldGetVisibleLines (const FieldType *fldP)
```

Parameters → *fldP*
Pointer to a field object (`FieldType` structure).

Returns Returns the number of lines the field displays. (This is the size of the `lines` array in the `FieldType` structure.)

See Also `FldGetNumberOfBlankLines()`, `FldCalcFieldHeight()`

FldGrabFocus Function

Purpose Turn the insertion point on (if the specified field is visible) and position the blinking insertion point in the field.

Declared In `Field.h`

Prototype

```
void FldGrabFocus (FieldType *fldP)
```

Parameters → *fldP*
Pointer to a field object (`FieldType` structure).

Returns Returns nothing.

Comments You rarely need to call this function directly. Instead, use `FrmSetFocus()`, which calls `FldGrabFocus` for you.

One instance where you need to call `FldGrabFocus` directly is to programmatically set the focus in a field that is contained in a table cell.

This function sets the field attribute `hasFocus` to `true`. (See `FieldAttrType`.)

See Also `FrmSetFocus()`, `FldReleaseFocus()`

FldHandleEvent Function

Purpose Handles events that affect the field, including the following: `keyDownEvent`, `penDownEvent`, and `fldEnterEvent`.

Declared In `Field.h`

Prototype

```
Boolean FldHandleEvent (FieldType *fldP,
    EventType *eventP)
```

Parameters → *fldP*
Pointer to a field object (`FieldType` structure).

→ *eventP*
Pointer to an event (`EventType` data structure).

Returns Returns `true` if the event was handled.

Comments When a `keyDownEvent` occurs in an editable text field, the keystroke appears in the field if it's a printable character or manipulates the insertion point if it's a "movement" character. The field is automatically updated.

When a `penDownEvent` occurs, the field sends a `fldEnterEvent` to the event queue.

When a `fldEnterEvent` occurs, the field grabs the focus. If the user has tapped twice in the current location, the word at that location is selected. If the user has tapped three times, the entire line is selected. Otherwise, the insertion point is placed in the specified position.

When a `menuCmdBarOpenEvent` occurs, the field adds paste, copy, cut, and undo buttons to the command toolbar. These buttons are only added if they make sense in the current context. That is, the cut button is only added if the field is editable, the paste button is only added if there is text on the clipboard and the field is editable, and the undo button is only added if there is an action to undo.

If the event alters the contents of the field, this function visually updates the field.

This function doesn't handle any events if the field is not editable or usable.

Compatibility Double-tapping to select a word and triple-tapping to select a line are only supported if 3.5 New Feature Set is present.

`FldHandleEvent` only handles the `menuCmdBarOpenEvent` if 3.5 New Feature Set is present.

FldInsert Function

Purpose Replace the current selection if any with the specified string and redraw the field.

Declared In `Field.h`

Prototype

```
Boolean FldInsert (FieldType *fldP,
   const Char *insertChars, UInt16 insertLen)
```

Parameters

→ *fldP*
Pointer to the field object (`FieldType` structure) to insert to.

→ *insertChars*
Text string to be inserted.

→ *insertLen*
Length in bytes of the text string to be inserted, not counting the trailing null character.

Returns Returns `true` if string was successfully inserted. Returns `false` if:

- The `insertLen` parameter is 0.
- The field is not editable.
- Adding the text would exceed the field's size limit (the `maxChars` value).
- More memory must be allocated for the field, and the allocation fails.

Comments If there is no current selection, the string passed is inserted at the position of the insertion point.

This function sets the field's dirty attribute and posts a `fldChangedEvent` to the event queue. If you call this function repeatedly, you may overflow the event queue with `fldChangedEvents`. An alternative is to remove the text handle from the field, change the text, and then set the field's handle again. See `FldGetTextHandle()` for a code example.

See Also `FldPaste()`, `FldDelete()`, `FldCut()`, `FldCopy()`

FldMakeFullyVisible Function

Purpose Generates an event to cause a dynamically resizable field to expand its height to make its text fully visible.

Declared In `Field.h`

Prototype

```
Boolean FldMakeFullyVisible (FieldType *fldP)
```

Parameters

→ *fldP*
Pointer to a field object (`FieldType` structure).

Returns Returns `true` if the field is dynamically resizable and was not fully visible; `false` otherwise.

Comments Use this function on a field whose `dynamicSize` attribute is `true` (see `FieldAttrType`).

This function does not actually resize the field. Instead, it computes how big the field should be to be fully visible and then posts this information to the event queue in a `fldHeightChangedEvent`.

NOTE: The event does not get generated if the number of lines in the field is equal to or greater than the value of the maximum lines attribute for the field.

If the field is contained in a table, the table's code handles the `fldHeightChangedEvent`. If the field is directly on a form, your application code should handle the `fldHeightChangedEvent` itself. The form code does not handle the event for you. Note that the constant `maxFieldLines` defines the maximum number of lines a field can expand to if the field is using the standard font.

See Also `TblHandleEvent()`

FldNewField Function

Purpose Create a new field object dynamically and install it in the specified form.

Declared In `Field.h`

Prototype

```
FieldType *FldNewField (void **formPP, UInt16 id,
   Coord x, Coord y, Coord width, Coord height,
   FontID font, UInt32 maxChars,
   Boolean editable,
   Boolean underlined, Boolean singleLine,
   Boolean dynamicSize,
   JustificationType justification,
   Boolean autoShift, Boolean hasScrollBar,
   Boolean numeric)
```

Parameters

↔ *formPP*
: Pointer to the pointer to the form in which the new field is installed. This value is not a handle; that is, the old form pointer value is not necessarily valid after this function returns. In subsequent calls, always use the new form pointer value returned by this function.

→ *id*
: Symbolic ID of the field, specified by the developer. By convention, this ID should match the resource ID (not mandatory).

→ *x*
: Horizontal coordinate of the upper-left corner of the field's boundaries, relative to the window in which it appears.

→ *y*
: Vertical coordinate of the upper-left corner of the field's boundaries, relative to the window in which it appears.

→ *width*
: Width of the field, expressed in pixels.

→ *height*
: Height of the field, expressed in pixels.

→ *font*
: Font to use to draw the field's text.

→ *maxChars*
Maximum number of bytes held by the field this function creates.

→ *editable*
Pass `true` to create a field in which the user can edit text. Pass `false` to create a field that cannot be edited.

→ *underlined*
Pass `noUnderline` for no underline, or `grayUnderline` to have the field underline the text it displays. On Palm OS® version 3.1 and higher, pass `solidUnderline` to use a solid underline instead of a dotted underline.

→ *singleLine*
Pass `true` to create a field that can display only a single line of text.

→ *dynamicSize*
Pass `true` to create a field that resizes dynamically according to the amount of text it displays.

→ *justification*
Pass either of the values `leftAlign` or `rightAlign` to specify left justification or right justification, respectively. The `centerAlign` value is not supported.

→ *autoShift*
Pass `true` to specify the use of Palm OS 2.0 (and later) auto-shift rules.

→ *hasScrollBar*
Pass `true` to attach a scroll bar control to the field this function creates.

→ *numeric*
Pass `true` to specify that only characters in the range of 0 through 9 are allowed in the field.

Returns Returns a pointer to the new field object or `NULL` if there wasn't enough memory to create the field. Out of memory situations could be caused by memory fragmentation.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FrmValidatePtr()`, `WinValidateHandle()`, `CtlValidatePointer()`, `FrmRemoveObject()`

FldPaste Function

Purpose Replace the current selection in the field, if any, with the contents of the text clipboard.

Declared In `Field.h`

Prototype

```
void FldPaste (FieldType *fldP)
```

Parameters

→ *fldP*
Pointer to a field object (`FieldType` structure).

Returns Returns nothing

Comments The function performs these actions:

- Scrolls the field, if necessary, so the insertion point is visible.
- Inserts the clipboard text at the position of the insertion point if there is no current selection.
- Positions the insertion point after the last character inserted.
- Doesn't delete the current selection if there is no text in the clipboard.

See Also `FldInsert()`, `FldDelete()`, `FldCut()`, `FldCopy()` `FldUndo()`

FldRecalculateField Function

Purpose Update the structure that contains the word-wrapping information for each visible line.

Declared In `Field.h`

Prototype

```
void FldRecalculateField (FieldType *fldP,
    Boolean redraw)
```

Parameters

→ *fldP*
Pointer to a field object (`FieldType` structure).

→ *redraw*
If `true`, redraws the field.

Returns Returns nothing.

Comments This function will allocate the memory block that contains the displayed lines information if, and only if, the block does not yet exist.

You should call this function when you change the field width or text length of the field. Do not call this function after changing the font or field height.

Note that many of the field functions, including `FldSetTextHandle()`, `FldInsert()`, and `FldDelete()`, recalculate the word-wrapping information for you.

Compatibility In releases prior to Palm OS 4.0, the word-wrapping information is only updated if the `redraw` parameter is set to `true`. As of Palm OS 4.0 it is updated whenever `FldRecalculateField` is called, regardless of the value of the `redraw` parameter.

FldReleaseFocus Function

Purpose Turn the blinking insertion point off if the field is visible and has the current focus, reset the Graffiti® or Graffiti 2 state, and reset the undo state.

Declared In `Field.h`

Prototype

```
void FldReleaseFocus (FieldType *fldP)
```

Parameters

→ *`fldP`*
Pointer to a field object (`FieldType` structure).

Returns Returns nothing.

Comments This function sets the field attribute `hasFocus` to `false`. (See `FieldAttrType`.)

Usually, you don't need to call this function. If the field is in a form or in a table that doesn't use custom drawing functions, the field code releases the focus for you when the focus changes to some other control. If your field is in any other type of object, such as a table that uses custom drawing functions or a gadget, you must call `FldReleaseFocus` when the focus moves away from the field.

See Also `FldGrabFocus()`

FldScrollable Function

Purpose Return `true` if the field is scrollable in the specified direction.

Declared In `Field.h`

Prototype

```
Boolean FldScrollable (const FieldType *fldP,
   WinDirectionType direction)
```

Parameters

→ *`fldP`*
Pointer to a field object (`FieldType` structure).

→ *`direction`*
The direction to test. `DirectionType` is defined in `Window.h`. It is an enum defining the constants `up` and `down`.

Returns Returns `true` if the field is scrollable in the specified direction; `false` otherwise.

See Also `FldScrollField()`

FldScrollField Function

Purpose Scroll a field up or down by the number of lines specified.

Declared In `Field.h`

Prototype

```
void FldScrollField (FieldType *fldP,
   UInt16 linesToScroll,
   WinDirectionType direction)
```

Parameters

→ *`fldP`*
Pointer to a field object (`FieldType` structure).

→ *`linesToScroll`*
Number of lines to scroll.

→ *`direction`*
The direction to scroll. `DirectionType` is defined in `Window.h`. It is an enum defining, among others, the constants `winUp` and `winDown`.

Returns Returns nothing.

Comments This function can't scroll horizontally, that is, right or left.

The field object is redrawn if it's scrolled; however, the scrollbar is not updated. Use SclSetScrollBar() to update the scrollbar. For example:

```
FldScrollField (fldP, linesToScroll,
direction);

// Update the scroll bar.
SclGetScrollBar (bar, &value, &min, &max,
  &pageSize);

if (direction == winUp)
  value -= linesToScroll;
else
  value += linesToScroll;

SclSetScrollBar (bar, value, min, max,
  pageSize);
```

If the field is not scrollable in the direction indicated, this function returns without performing any work. You can use FldScrollable() before calling this function to see if the field can be scrolled.

See Also FldScrollable(), FldSetScrollPosition()

FldSendChangeNotification Function

Purpose Send a fldChangedEvent to the event queue.

Declared In `Field.h`

Prototype

```
void FldSendChangeNotification
    (const FieldType *fldP)
```

Parameters → *fldP*
Pointer to a field object.

Returns Returns nothing.

Comments This function is used internally by the field code. You normally never call it in application code.

FldSendHeightChangeNotification Function

Purpose Send a `fldHeightChangedEvent` to the event queue.

Declared In `Field.h`

Prototype

```
void FldSendHeightChangeNotification
    (const FieldType *fldP, UInt16 pos,
    Int16 numLines)
```

Parameters

→ *fldP*
Pointer to a field object.

→ *pos*
Character position of the insertion point.

→ *numLines*
New number of lines in the field.

Returns Returns nothing.

Comments This function is used internally by the field code. You normally never call it in application code.

FldSetAttributes Function

Purpose Set the attributes of a field.

Declared In `Field.h`

Prototype

```
void FldSetAttributes (FieldType *fldP,
    const FieldAttrType *attrP)
```

Parameters

→ *fldP*
Pointer to a `FieldType` structure.

→ *attrP*
Pointer to the attributes.

Returns Returns nothing.

Comments This function does not do anything to make the new attribute values take effect. For example, if you use this function to change the value of the underline attribute, you won't see its effect until you call `FldDrawField()`.

You usually do not have to modify field attributes at runtime, so you rarely need to call this function.

WARNING! You must not call this function to change any attributes that are noted as "for internal use only."

The proper way to use `FldSetAttributes` is to:

1. Call `FldGetAttributes` to retrieve the attributes.
2. Set the specific flags that you want to modify.
3. Call `FldSetAttributes` to make the modifications.

WARNING! You must not call any field routines between calling `FldGetAttributes` and `FldSetAttributes`; this can cause the attributes to be out of sync, with unpredictable results.

See Also `FldGetAttributes()`, `FieldAttrType`

FldSetBounds Function

Purpose Change the position or size of a field.

Declared In `Field.h`

Prototype

```
void FldSetBounds (FieldType *fldP,
    const RectangleType *rP)
```

Parameters

→ *fldP*
Pointer to a field object (`FieldType` structure).

→ *rP*
Pointer to a RectangleType structure that contains the new bounds of the display.

Returns Returns nothing. May raise a fatal error message if the memory block that contains the word-wrapping information needs to be resized and there is not enough space to do so.

Comments If the field is visible, the field is redrawn within its new bounds.

NOTE: You can change the height or location of the field while it's visible, but do not change the width.

The memory block that contains the word-wrapping information (see `LineInfoType`) will be resized if the number of visible lines is

changed. The insertion point is assumed to be off when this routine is called.

Make sure that `rect` is at least as tall as a single line in the current font. (You can determine this value by calling `FntLineHeight()`.) If it's not, results are unpredictable.

See Also `FldGetBounds()`, `FrmSetObjectBounds()`

FldSetDirty Function

Purpose Set whether the field has been modified.

Declared In `Field.h`

Prototype

```
void FldSetDirty (FieldType *fldP, Boolean dirty)
```

Parameters

→ *fldP*
Pointer to a field object (`FieldType` structure).

→ *dirty*
`true` if the text is modified.

Returns Returns nothing.

Comments You typically call this function when you want to clear the dirty attribute. The dirty attribute is set when the user enters or deletes text in the field. It is also set by certain field functions, such as `FldInsert()` and `FldDelete()`.

See Also `FldDirty()`

FldSetFont Function

Purpose Set the font used by the field, update the word-wrapping information, and draw the field if the field is visible.

Declared In `Field.h`

Prototype

```
void FldSetFont (FieldType *fldP, FontID fontID)
```

Parameters

→ *fldP*
Pointer to a field object (`FieldType` structure).

→ *fontID*
ID of new font.

Returns Returns nothing.

See Also `FldGetFont()`, `FieldAttrType`

FldSetInsertionPoint Function

Purpose Set the location of the insertion point based on a specified string position.

Declared In `Field.h`

Prototype

```
void FldSetInsertionPoint (FieldType *fldP,
    UInt16 pos)
```

Parameters

→ *fldP*
Pointer to a `FieldType` structure.

→ *pos*
New location of the insertion point, given as a valid offset in bytes into the field's text. On systems that support multi-byte characters, you must make sure that this specifies an inter-character boundary (does not specify the middle or end bytes of a multi-byte character).

Returns Nothing.

Comments This routine differs from `FldSetInsPtPosition()` in that it doesn't make the character position visible. `FldSetInsertionPoint` also doesn't make the field the current focus of input if it was not already.

If `pos` indicates a position beyond the end of the text in the field, the insertion point is set to the end of the field's text.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `TxtCharBounds()`

FldSetInsPtPosition Function

Purpose Set the location of the insertion point for a given string position.

Declared In `Field.h`

Prototype

```
void FldSetInsPtPosition (FieldType *fldP,
    UInt16 pos)
```

Parameters

→ *fldP*
Pointer to a field object (`FieldType` structure).

→ *pos*
New location of the insertion point, given as a valid offset in bytes into the field's text. On systems that support multi-byte characters, you must make sure that this specifies an inter-character boundary (does not specify the middle or end bytes of a multi-byte character).

Returns Returns nothing.

Comments If the position is beyond the visible text, the field is scrolled until the position is visible.

See Also `FldGetInsPtPosition()`, `TxtCharBounds()`

FldSetMaxChars Function

Purpose Set the maximum number of bytes the field accepts (the `maxChars` value).

Declared In `Field.h`

Prototype

```
void FldSetMaxChars (FieldType *fldP,
    UInt16 maxChars)
```

Parameters

→ *fldP*
Pointer to a field object (`FieldType` structure).

→ *maxChars*
Maximum size in bytes of the characters the user may enter. You may specify any value up to `maxFieldTextLen`.

Returns Returns nothing.

Comments Line feed characters are counted when the length of characters is determined.

See Also `FldGetMaxChars()`

FldSetMaxVisibleLines Function

Purpose Allows the creation of tables and fields smaller than 121 pixels tall that still drag-select when there are more lines of text than will fit in the space provided.

Declared In `Field.h`

Prototype

```
void FldSetMaxVisibleLines (FieldType *fldP,
    UInt8 maxLines)
```

Parameters

→ *fldP*
Pointer to a field object (`FieldType` structure).

→ *maxLines*
Maximum number of lines to which the field will visually grow.

Returns Returns nothing.

Comments A field can be dynamically expandable. When it is, the field package needs to know the maximum number of lines that should be visible so it can prevent the field from being expanded further. Since field expansion is actually handled by enclosing objects—tables or forms—this function's primary purpose is to allow the enclosing object to tell the field how big it can get.

By default, tables assume that the field can get as big as the table.

If you don't call this function, fields expect to be at least 121 pixels tall and try to grow repeatedly until they are.

FldSetScrollPosition Function

Purpose Scroll the field such that the character at the indicated offset is the first character on the first visible line. Redraw the field if necessary.

Declared In `Field.h`

Prototype

```
void FldSetScrollPosition (FieldType *fldP,
    UInt16 pos)
```

Parameters

→ *fldP*
Pointer to a field object (`FieldType` structure).

→ *pos*
Byte offset into the field's text string of first character to be made visible. On systems that support multi-byte characters, you must make sure that this specifies an inter-character boundary (does not specify the middle or end bytes of a multi-byte character).

Returns Returns nothing.

Comments This function scrolls the field but does not update the field's scrollbar. You should update the scrollbar after calling this function. To do so, first call `FldGetScrollValues()` to determine the values to use, and then call `SclSetScrollBar()`.

See Also `FldGetScrollPosition()`, `FldScrollField()`, `TxtCharBounds()`

FldSetSelection Function

Purpose Set the current selection in a field and highlight the selection if the field is visible.

Declared In `Field.h`

Prototype

```
void FldSetSelection (FieldType *fldP,
   UInt16 startPosition, UInt16 endPosition)
```

Parameters

→ *fldP*
Pointer to a field object (`FieldType` structure).

→ *startPosition*
Starting offset of the character range to highlight, given as a byte offset into the field's text.

→ *endPosition*
Ending offset of the character range to highlight. The ending offset should be greater than or equal to the starting offset. On systems that support multi-byte characters, this position must be an inter-character boundary. That is, it must not point to a middle byte of a multi-byte character.

Returns Returns nothing.

Comments To cancel a selection, set both `startPosition` and `endPosition` to the same value. If `startPosition` equals `endPosition`, then the current selection is unhighlighted.

If either `startPosition` or `endPosition` point to an intra-character boundary, `FldSetSelection` attempts to move that offset backward, toward the beginning of the string, until the offset points to an inter-character boundary (i.e., the start of a character).

See Also `TxtCharBounds()`

FldSetText Function

Purpose Set the text value of the field without updating the display.

Declared In `Field.h`

Prototype

```
void FldSetText (FieldType *fldP,
   MemHandle textHandle, UInt16 offset,
   UInt16 size)
```

Parameters

→ *fldP*
Pointer to a field object (`FieldType` structure).

→ *textHandle*
Unlocked handle of a block containing a null-terminated text string. Pass `NULL` for this parameter to remove the association between the field and the string it is currently displaying so that the string is not freed with the field when the form is deleted.

→ *offset*
Offset from start of block to start of the text string.

→ *size*
The allocated size of the text string. This is not the string length, and should not be set to `0`, unless you are setting the text to the empty string.

Returns Returns nothing.

Comments This function allows applications to perform editing in place in memory. You can use it to point the field to a string in a database record so that you can edit that string directly using field routines.

As characters are added to the field's text, the block that contains the text is grown. So that the block doesn't have to be expanded for each character, it is expanded several bytes at a time; this expansion may result in some unused space in the text block. As characters are removed from the field’s text, the space is not automatically

reclaimed. Because adding or removing characters when editing a data record in place may result in unused space at the end of the field's text block, applications should call `FldCompactText()` on before the field is unlocked to release any unused space.

The handle that you pass to this function is assumed to contain a null-terminated string starting at `offset` bytes in the memory chunk. The string should be between 0 and `size` - 1 bytes in length. The field does not make a copy of the memory chunk or the string data; instead, it stores the handle to the record in its structure.

WARNING! You cannot use this function to set two fields on a form so that they share a single string value. Thus, for instance, if you have a single string containing a person's name you cannot call `FldSetText` twice with the same string (but a different offset) to set up a first name field and a last name field.

`FldSetText` updates the word-wrapping information and places the insertion point after the last visible character, but it does not update the display. You must call `FldDrawField()` after calling this function to update the display.

`FldSetText` increments the lock count for `textHandle` and decrements the lock count of its previous text handle (if any).

Because `FldSetText` (and `FldSetTextHandle`) may be used to edit database records, they do not free the memory associated with the previous text handle. If the previous text handle points to a string on the dynamic heap and you want to free it, use `FldGetTextHandle()` to obtain the handle before using `FldSetText` and then free that handle after using `FldSetText`. (See `FldSetTextHandle()` for a code example.)

If the field points to a database record, you want the memory associated with the text handle to persist; however, this memory and all other memory associated with the field is freed when the field itself is freed, which happens when the form is closed. If you don't want the memory associated with the text handle freed when the field is freed, use `FldSetText` and pass `NULL` for the text handle immediately before the form is closed. Passing `NULL` removes the association between the field and the text handle that you want retained. That text handle is unlocked as a result of the

`FldSetText` call, and when the field is freed, there is no text handle to free with it.

See Also `FldSetTextPtr()`, `FldSetTextHandle()`

FldSetTextAllocatedSize Function

Purpose Set the number of bytes allocated to hold the field's text string. Don't confuse this with the actual length of the text string displayed in the field.

Declared In `Field.h`

Prototype

```
void FldSetTextAllocatedSize (FieldType *fldP,
    UInt16 allocatedSize)
```

Parameters

→ *fldP*
Pointer to a field object (`FieldType` structure).

→ *allocatedSize*
Number of bytes to allocate for the text.

Returns Returns nothing.

Comments This function generally is not used. It does not resize the field's allocated memory for the text string; it merely sets the `textBlockSize` field of the `FieldType` structure. The value of this field is computed and maintained internally by the field, so you should not have to call `FldSetTextAllocatedSize` directly.

See Also `FldGetTextAllocatedSize()`, `FldCompactText()`

FldSetTextHandle Function

Purpose Set the text value of a field to the string associated with the specified handle. Does not update the display.

Declared In `Field.h`

Prototype

```
void FldSetTextHandle (FieldType *fldP,
    MemHandle textHandle)
```

Parameters

→ *fldP*
Pointer to a field object (`FieldType` structure).

→ *textHandle*
Unlocked handle of a field's text string. Pass `NULL` for this parameter to remove the association between the field and the string it is currently displaying so that the string is not freed with the field when the form is deleted.

Returns Returns nothing.

Comments This function differs from `FldSetText()` in that it uses the entire memory chunk pointed to by `textHandle` for the string. In fact, this function simply calls `FldSetText` with an offset of 0 and a size equal to the entire length of the memory chunk. Use it to have the field edit a string in a database record if the entire record consists of that string, or use it to have the field edit a string in the dynamic heap.

As characters are added to the field's text, the block that contains the text is grown. So that the block doesn't have to be expanded for each character, it is expanded several bytes at a time; this expansion may result in some unused space in the text block. As characters are removed from the field's text, the space is not automatically reclaimed. Because adding or removing characters when editing a data record in place may result in unused space at the end of the field's text block, applications should call `FldCompactText()` on before the field is unlocked to release any unused space.

`FldSetTextHandle` updates the word-wrapping information and places the insertion point after the last visible character, but it does not update the display. You must call `FldDrawField()` after calling this function to update the display.

`FldSetTextHandle` increments the lock count for `textHandle` and decrements the lock count of its previous text handle (if any).

Because `FldSetTextHandle` (and `FldSetText`) may be used to edit database records, they do not free the memory associated with the previous text handle. If the previous text handle points to a string on the dynamic heap and you want to free it, use `FldGetTextHandle()` to obtain the handle before using `FldSetText` and then free that handle after using `FldSetText`. For example:

```
/* get the old text handle */
oldTxtH = FldGetTextHandle(fldP);
```

```
/* change the text and update the display */
FldSetTextHandle(fldP, txtH);
FldDrawField(fldP);

/* free the old text handle */
if (oldTxtH != NULL)
  MemHandleFree(oldTxtH);
```

If the field points to a database record, you want the memory associated with the text handle to persist; however, this memory and all other memory associated with the field is freed when the field itself is freed, which happens when the form is closed. If you don't want the memory associated with the text handle freed when the field is freed, use `FldSetTextHandle` and pass `NULL` for the text handle immediately before the form is closed. Passing `NULL` removes the association between the field and the text handle that you want retained. That text handle is unlocked as a result of the `FldSetTextHandle` call, and when the field is freed, there is no text handle to free with it.

See Also `FldSetTextPtr()`, `FldSetText()`

FldSetTextPtr Function

Purpose Set a noneditable field's text to point to the specified text string.

Declared In `Field.h`

Prototype

```
void FldSetTextPtr (FieldType *fldP, Char *textP)
```

Parameters

→ *fldP*
Pointer to a field object (`FieldType` structure).

→ *textP*
Pointer to a null-terminated string.

Returns Returns nothing. May display an error message if passed an editable text field.

Comments Do not call `FldSetTextPtr` with an editable text field. Instead, call `FldSetTextHandle()` for editable text fields. `FldSetTextPtr` is intended for displaying noneditable text in the user interface.

If the field has more than one line, use `FldRecalculateField()` to recalculate the word wrapping.

This function does *not* visually update the field. Use `FldDrawField()` to do so.

The field never frees the string that you pass to this function, even when the field itself is freed. You must free the string yourself. Before you free the string, make sure the field is not still displaying it. Set the field's string pointer to some other string or call `FldSetTextPtr(fldP, NULL)` before freeing a string you have passed using this function.

Compatibility On versions of Palm OS prior to Palm OS 4.0 you cannot supply `NULL` for the *`textP`* parameter: the function calls `StrLen()` on the `NULL` pointer which results in a crash.

See Also `FldSetTextHandle()`, `FldGetTextPtr()`

FldSetUsable Function

Purpose Set a field to usable or nonusable.

Declared In `Field.h`

Prototype

```
void FldSetUsable (FieldType *fldP,
   Boolean usable)
```

Parameters

`fldP`
Pointer to a `FieldType` structure.

`usable`
`true` to set usable; false to set nonusable.

Returns Returns nothing.

Comments A nonusable field doesn't display or accept input.

Use `FrmHideObject()` and `FrmShowObject()` instead of using this function.

See Also `FldEraseField()`, `FldDrawField()`, `FieldAttrType`

FldUndo Function

Purpose Undo the last change made to the field object, if any. Changes include typing, backspaces, delete, paste, and cut.

Declared In `Field.h`

Prototype

```
void FldUndo (FieldType *fldP)
```

Parameters

`fldP`
Pointer to the field (`FieldType` structure) that has the focus.

Returns Returns nothing.

See Also `FldPaste()`, `FldCut()`, `FldDelete()`, `FldInsert()`

FldWordWrap Function

Purpose Given a string and a width, return the number of bytes of characters that can be displayed using the current font.

Declared In `Field.h`

Prototype

```
UInt16 FldWordWrap (const Char *chars,
    Int16 maxWidth)
```

Parameters

→ *chars*
Pointer to a null-terminated string.

→ *maxWidth*
Maximum line width in pixels.

Returns Returns the length in bytes of the characters that can be displayed.

See Also `FntWordWrap()`

10

Find

This chapter describes the global find facility API declared in the header file `Find.h`.

Find Functions

FindDrawHeader Function

Purpose Draw the header line that separates, by application, the list of found items.

Declared In `Find.h`

Prototype

```
Boolean FindDrawHeader (FindParamsPtr findParams,
    Char const *title)
```

Parameters

→ *findParams*
Pointer to the `sysAppLaunchCmdFind` launch code's parameter block.

→ *title*
String to display as the title for the current application.

Returns Returns `true` if Find screen is filled up. Applications should exit from the search if this occurs.

Comments Call this function once at the beginning of your application's response to the `sysAppLaunchCmdFind` launch code. This function draws a header for your application's Find results. The header separates the search results from your application with the search results from another application.

If your application searches multiple databases, you may also use `FindDrawHeader` as a separator between databases.

FindGetLineBounds Function

Purpose Returns the bounds of the next available line for displaying a match in the Find results dialog.

Declared In `Find.h`

Prototype

```
void FindGetLineBounds
    (const FindParamsType *findParams,
    RectanglePtr r)
```

Parameters

→ *findParams*
Pointer to the `sysAppLaunchCmdFind` launch code's parameter block.

← *r*
The bounds of the area that should contain the next line of results.

Returns Returns nothing.

FindSaveMatch Function

Purpose Saves the record and position within the record of a text search match. This information is saved so that it's possible to later navigate to the match.

Declared In `Find.h`

Prototype

```
Boolean FindSaveMatch (FindParamsPtr findParams,
    UInt16 recordNum, UInt16 pos, UInt16 fieldNum,
    UInt32 appCustom, UInt16 cardNo, LocalID dbID)
```

Parameters

→ *findParams*
Pointer to the `sysAppLaunchCmdFind` launch code's parameter block.

→ *recordNum*
Record index. This parameter sets the `recordNum` field in the `sysAppLaunchCmdGoTo`'s parameter block.

→ *pos*
Offset of the match string from start of record. This parameter sets the `matchPos` field in the `sysAppLaunchCmdGoTo`'s parameter block.

→ *fieldNum*
Field number that the string was found in. This parameter sets the `matchFieldNum` field in the `sysAppLaunchCmdGoTo`'s parameter block.

→ *appCustom*
Extra data the application can save with a match. This parameter sets the `matchCustom` field in the `sysAppLaunchCmdGoTo`'s parameter block.

→ *cardNo*
Card number of the database that contains the match. This parameter sets the `dbCardNo` field in the `sysAppLaunchCmdGoTo`'s parameter block.

→ *dbID*
Local ID of the database that contains the match. This parameter sets the `dbID` field in the `sysAppLaunchCmdGoTo`'s parameter block.

Returns Returns `true` if Find screen is filled up. Applications should exit from the search if this occurs.

Comments Call this function when your application finds a record with a matching string (`FindStrInStr()` or `TxtFindString()` returns `true`). This function saves the information you pass. If the user clicks this selection in the Find results dialog, the information is retrieved and used to set up the `sysAppLaunchCmdGoTo` launch code's parameter block.

You can use the `appCustom` field for any application-specific data that might be needed to navigate to the record if the user selects it. It's common for localizable applications to set `appCustom` to the length of the matching string because the global find facility cannot correctly determine the length of the matching string on systems with multi-byte character sets. In some character encodings, one character may be accurately represented as either a single-byte character or a multi-byte character. The `TxtFindString` function accurately matches single-byte characters against their multi-byte equivalents and returns the length of the matching string. If you pass `TxtFindString`'s return value as the `appCustom` parameter to `FindSaveMatch`, the `matchCustom` field of the `sysAppLaunchCmdGoTo` parameter block contains the length of the matching string.

If your application requires more custom information, you can store the information in a feature and store the feature number in the `appCustom` field. See the "Feature Manager" chapter for more information.

FindStrInStr Function

Purpose Perform a case-blind prefix search for a string in another string. This function assumes that the string to find has already been normalized for searching.

Declared In `Find.h`

Prototype

```
Boolean FindStrInStr (Char const *strToSearch,
   Char const *strToFind, UInt16 *posP)
```

Parameters

→ *strToSearch*
: String to search.

→ *strToFind*
: Normalized version of the text string to be found.

← *posP*
: If a match is found, contains the offset of the match within `strToSearch`.

Returns Returns `true` if the string was found. `FindStrInStr` matches the beginnings of words only; that is, `strToFind` must be a prefix of one of the words in `strToSearch` for `FindStrInStr` to return `true`.

Comments Don't use this function on systems that support the text manager. Instead, use `TxtFindString()`, which performs searches on strings that contain multi-byte characters and returns the length of the matching text.

For backward compatibility with systems that don't support the text manager, use `TxtGlueFindString`, found in the PalmOSGlue library. `TxtGlueFindString` calls `TxtFindString` if the text manager is present, or `FindStrInStr` if it is not present. For more information, see Chapter 80, "PalmOSGlue Library."

The method by which a search string is normalized varies depending on the version of Palm OS® and the character encoding supported by the device. The string passed to your application in

the `strToFind` field of the `sysAppLaunchCmdFind` launch code parameter block has already been normalized. It can be passed directly to `FindStrInStr`, `TxtFindString`, or `TxtGlueFindString`. If you need to create your own normalized search string, use `TxtGluePrepFindString`, also in the PalmOSGlue library.

11

Forms

This chapter provides the following information about form objects:

Form Structures and Types 259
Form Constants 279
Form Resources 289
Form Functions 290
Application-Defined Functions 345

The header file `Form.h` declares the API that this chapter describes. For more information on forms, see the section "Text" in the *Palm OS Programmer's Companion*, vol. I.

Form Structures and Types

AlertTemplateType Struct

Purpose Structure of an alert resource. Then operating system uses this structure when it needs to retrieve an alert resource from a resource database and manipulate the alert's contents prior to display.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `AlertTemplateType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Declared In `Form.h`

Prototype

```
typedef struct AlertTemplateTag {
   UInt16 alertType;
   UInt16 helpRscID;
   UInt16 numButtons;
   UInt16 defaultButton;
} AlertTemplateType
```

Fields

`alertType`
: The alert type. One of the values defined under `AlertType`.

`helpRscID`
: Symbolic ID of the resource that provides the alert's online help.

`numButtons`
: The number of buttons on the alert.

`defaultButton`
: Symbolic ID of the button that provides the alert's default action.

Comments This structure is immediately followed by the alert title, the alert message, and the button strings, each null-terminated and each immediately following the previous.

FormActiveStateType Struct

Purpose Structure that is passed to `FrmSaveActiveState()` and `FrmRestoreActiveState()` when saving and restoring the active form and window state.

Declared In `Form.h`

Prototype

```
typedef struct FormActiveStateType {
   UInt16 data[11];
} FormActiveStateType
```

Fields

`data`
: Abstract data containing the form and window state information. Note that this data is abstract since the structure's contents can differ significantly from one release of Palm OS to another. Applications should not attempt to read or write the contents of this field.

Compatibility This structure was added in Palm OS 3.0.

FormAttrType Struct

Purpose The `FormAttrType` bit field defines the visible characteristics of the form.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `FormAttrType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  UInt16 usable:1;
  UInt16 enabled:1;
  UInt16 visible:1;
  UInt16 dirty:1;
  UInt16 saveBehind:1;
  UInt16 graffitiShift:1;
  UInt16 globalsAvailable:1;
  UInt16 doingDialog:1;
  UInt16 exitDialog:1;
  UInt16 reserved:7;
  UInt16 reserved2;
} FormAttrType;
```

Your code should treat the `FormAttrType` bit field as opaque. Do not attempt to change bit field member values directly.

Fields

`usable`
: Not set if the form is not considered part of the current interface of the application, and it doesn't appear on screen.

`enabled`
: Not used.

`visible`
: Set or cleared internally when the field object is drawn or erased.

`dirty`
: Not used.

`saveBehind`
: Set if the bits behind the form are saved when the form is drawn.

`graffitiShift`
: Set if the shift indicator is supported.

`globalsAvailable`
: System use only.

`doingDialog`
: System use only.

`exitDialog`
: System use only.

`reserved`
: Reserved for system use.

`reserved2`
: Reserved for system use.

Compatibility The `globalsAvailable`, `doingDialog`, and `exitDialog` flags are present only if 3.5 New Feature Set is present.

FormBitmapType Struct

Purpose The `FormBitmapType` structure defines the visible characteristics of a bitmap on a form.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `FormBitmapType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  FormObjAttrType attr;
  PointType pos;
  UInt16 rscID;
} FormBitmapType;
```

Fields

`attr`
: See `FormObjAttrType()`.

`pos`
: Location of the bitmap.

`rscID`
: Resource ID of the bitmap. If you use `DmGetResource()` with this value as the resource ID, it returns a pointer to a `BitmapType` structure.

FormFrameType Struct

Purpose The `FormFrameType` structure defines a frame that appears on the form.

Prototype

```
typedef struct {
  UInt16 id;
  FormObjAttrType attr;
  RectangleType rect;
  UInt16 frameType;
} FormFrameType;
```

Fields

`id`
: ID of the frame.

`attr`
: See `FormObjAttrType`.

`rect`
: Location and size of the frame.

`frameType`
: The type of frame.

FormGadgetAttrType Struct

Purpose The `FormGadgetAttrType` bit field defines a gadget's attributes.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `FormGadgetAttrType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  UInt16 usable:1;
  UInt16 extended:1;
  UInt16 visible:1;
  UInt16 reserved:13;
} FormGadgetAttrType;
```

Your code should treat the `FormGadgetAttrType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

Fields

`usable`
: Not set if the gadget is not considered part of the current interface of the application, and it doesn't appear on screen. This is set by `FrmShowObject()` and cleared by `FrmHideObject()`.

`extended`
: If set, the gadget is an extended gadget. Extended gadgets are supported if 3.5 New Feature Set is present. An extended gadget has the `handler` field defined in its `FormGadgetType`. If not set, the gadgets is a standard gadget compatible with all releases of Palm OS®.

`visible`
: Set or cleared when the gadget is drawn or erased. `FrmHideObject()` clears this value. You should set it explicitly in the gadget's callback function (if it has one) in response to a draw request.

`reserved`
: Reserved for future use.

Many form functions (`FrmGetObjectType()`, `FrmHideObject()`, and `FrmGetObjectBounds()`, for example) take an object index as one of their arguments. The most common way to get an object's index is to call `FrmGetObjectIndex()`. `FrmGetObjectIndex` takes a form ID and returns the form object's index. This is the routine one should use in most cases, because the application usually knows the object ID. However, gadgets and specifically extended gadgets, have APIs with callbacks that pass back the gadget pointer and not the ID. In those cases, the only way to get the object index (so one can use the `FrmGetObject*` APIs) is to use the function `FrmGetObjectIndexFromPtr()`.

If you need the same functionality on pre-Palm OS 4.0 systems then you can accomplish the same thing with the following code snippet.

```
UInt16 index;
UInt16 objIndex = frmInvalidObjectId;
UInt16 numObjects = FrmGetNumberOfObjects(frmP)
for (index = 0; index < numObjects; index++) {
     if (FrmGetObjectPtr(index) == myObjPtr) {
          // Found it
          objIndex = index;
          break;
    }
}
```

Compatibility This type is defined only if 3.5 New Feature Set is present.

FormGadgetType Struct

Purpose The `FormGadgetType` structure defines a gadget object that appears on a form.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `FormGadgetType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  UInt16 id;
  FormGadgetAttrType attr;
  RectangleType rect;
  const void *data;
  FormGadgetHandlerType *handler;
} FormGadgetType;
```

Your code should treat the `FormGadgetType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

Fields

`id`
: ID of the gadget resource.

`attr`
: See `FormGadgetAttrType`.

`rect`
: Location and size of the object.

`data`
: Pointer to any specific data that needs to be stored. You can set and retrieve the value of this field with `FrmGetGadgetData()` and `FrmSetGadgetData()`.

`handler`
: Pointer to a callback function that controls the gadget's behavior and responds to events. You can set this field with `FrmSetGadgetHandler()`.

Many form functions (`FrmGetObjectType()`, `FrmHideObject()`, and `FrmGetObjectBounds()`, for example) take an object index as one of their arguments. The most common way to get an object's index is to call `FrmGetObjectIndex()`. `FrmGetObjectIndex` takes a form ID and returns the form object's index. This is the routine one should use in most cases, because the application usually knows the object ID. However, gadgets have APIs with callbacks that pass back the gadget pointer and not the ID. In those cases, the only way to get the object index (so one can use the `FrmGetObject*` APIs) is to use the function `FrmGetObjectIndexFromPtr()`.

If you need the same functionality on pre-Palm OS 4.0 systems then you can accomplish the same thing with the following code snippet.

```
UInt16 index;
UInt16 objIndex = frmInvalidObjectId;
UInt16 numObjects = FrmGetNumberOfObjects(frmP)
for (index = 0; index < numObjects; index++) {
     if (FrmGetObjectPtr(index) == myObjPtr) {
          // Found it
          objIndex = index;
          break;
     }
}
```

Compatibility

In Palm OS® releases prior to 3.5, the `attr` field was of type `FormObjAttrType()` and the `handler` field did not exist.

FormGadgetTypeInCallback Struct

Purpose

The `FormGadgetTypeInCallback` structure is passed to your extended gadget handler and is identical to `FormGadgetType` except that its contents are *not* hidden when DO_NOT_ALLOW_ACCESS_TO_INTERNALS_OF_STRUCTS is defined. This allows you to freely access the contents of an extended gadget structure from within your extended gadget callback functions.

Prototype

```
typedef struct {
  UInt16 id;
  FormGadgetAttrType attr;
  RectangleType rect;
  const void *data;
  FormGadgetHandlerType *handler;
} FormGadgetTypeInCallback;
```

Fields

`id`
: ID of the gadget resource.

`attr`
: See `FormGadgetAttrType`.

`rect`
: Location and size of the object.

`data`
: Pointer to any specific data that needs to be stored.

`handler`
: Pointer to a callback function that controls the gadget's behavior and responds to events.

Many form functions (`FrmGetObjectType()`, `FrmHideObject()`, and `FrmGetObjectBounds()`, for example) take an object index as one of their arguments. The most common way to get an object's index is to call `FrmGetObjectIndex()`. `FrmGetObjectIndex` takes a form ID and returns the form object's index. This is the routine one should use in most cases, because the application usually knows the object ID. However, extended gadgets have APIs with callbacks that pass back the gadget pointer and not the ID. In those cases, the only way to get the object index (so one can use the `FrmGetObject*` APIs) is to use the function `FrmGetObjectIndexFromPtr()`.

If you need the same functionality on pre-Palm OS 4.0 systems then you can accomplish the same thing with the following code snippet.

```
UInt16 index;
UInt16 objIndex = frmInvalidObjectId;
UInt16 numObjects = FrmGetNumberOfObjects(frmP)
for (index = 0; index < numObjects; index++) {
     if (FrmGetObjectPtr(index) == myObjPtr) {
          // Found it
          objIndex = index;
          break;
     }
}
```

Compatibility Introduced in the Palm OS 4.0 SDK Update 1.

FormLabelType Struct

The `FormLabelType` structure defines a label that appears on a form.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `FormLabelType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

```
typedef struct {
  UInt16 id;
  PointType pos;
  FormObjAttrType attr;
  FontID fontID;
  UInt8 reserved;
  Char *text;
} FormLabelType;
```

Your code should treat the `FormLabelType` structure as opaque. Do not attempt to change structure member values directly.

Fields

`id`
: Resource ID of the label.

`pos`
: Location of the label.

`attr`
: See `FormObjAttrType`.

`fontID`
: Font ID of the font used for the label.

`reserved`
: Reserved for future use.

`text`
: Text of the label.

FormLineType Struct

Purpose The `FormLineType` structure defines a line appearing on a form.

Prototype

```
typedef struct {
  FormObjAttrType attr;
  PointType point1;
  PointType point2;
} FormLineType;
```

Your code should treat the `FormLineType` structure as opaque. Do not attempt to change structure member values directly.

Fields

`attr`
: See `FormObjAttrType`.

`point1`
: Starting point of the line.

`point2`
: Ending point of the line.

FormObjAttrType Struct

Purpose The `FormObjAttrType` bit field defines a form object's attributes.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `FormObjAttrType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  UInt16 usable : 1;
  UInt16 reserved : 15;
} FormObjAttrType;
```

Your code should treat the `FormObjAttrType` structure as opaque. Do not attempt to change structure member values directly.

Fields

`usable`
: Not set if the object is not considered part of the current interface of the application, and it doesn't appear on screen.

`reserved`
: Reserved for future use.

FormObjectType Struct

Purpose

The `FormObjectType` union points to the C structure for a user interface object that appears on the form.

Prototype

```
typedef union {
  void *ptr;
  FieldType *field;
  ControlType *control;
  GraphicControlType *graphicControl;
  SliderControlType *sliderControl;
  ListType *list;
  TableType *table;
  FormBitmapType *bitmap;
  FormLabelType *label;
  FormTitleType *title;
  FormPopupType *popup;
  FormGraffitiStateType *grfState;
  FormGadgetType *gadget;
  ScrollBarType *scrollBar;
} FormObjectType;
```

Your code should treat the `FormObjectType` structure as opaque. Do not attempt to change structure member values directly.

Fields

`ptr`
: Used when the object's type is not one of those specified below.

`field`
: Text field's structure. See `FieldType`.

`control`
: Control's structure. See `ControlType`.

`graphicControl`
: Graphic button structure. See `GraphicControlType`.

`sliderControl`
: Slider control structure. See `SliderControlType`.

`list`
: List object's structure. See `ListType`.

`table`
: Table structure. See `TableType`.

`bitmap`
: Form bitmap's structure. See `FormBitmapType`.

`label`
: Label's structure. See `FormLabelType`.

`title`
: Form title's structure. See `FormTitleType`.

`popup`
: Popup list's structure. See `FormPopupType`.

`grfState`
: Shift indicator's structure. See `FrmGraffitiStateType`.

`gadget`
: Gadget (custom UI resource) structure. See `FormGadgetType`.

`scrollbar`
: Scroll bar's structure. See `ScrollBarType`.

Compatibility

The `graphicControl` and `sliderControl` fields are only defined if 3.5 New Feature Set is present.

FormObjListType Struct

Purpose The `FormObjectListType` structure specifies a user interface object that appears on the form.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `FormObjListType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  FormObjectKind objectType;
  UInt8 reserved;
  FormObjectType object;
} FormObjListType;
```

Your code should treat the `FormObjListType` structure as opaque. Do not attempt to change structure member values directly.

Fields

`objectType`
: Specifies the type of the object (control, field, etc.). See `FormObjectKind`.

`reserved`
: Reserved for future use.

`object`
: The C data structure that defines the object. See `FormObjectType`.

FormPopupType Struct

Purpose The `FormPopupType` structure defines a popup list that appears on a form.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `FormPopupType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  UInt16 controlID;
  UInt16 listID;
} FormPopupType;
```

Your code should treat the `FormPopupType` structure as opaque. Do not attempt to change structure member values directly.

Fields

`controlID`
: Resource ID of the popup trigger control that triggers the list's display.

`listID`
: Resource ID of the list object that defines the popup list.

FormPtr Typedef

Purpose

The `FormPtr` type defines a pointer to a `FormType` structure.

Prototype

```
typedef FormType *FormPtr;
```

FormRectangleType Struct

Purpose

The `FormRectangleType` structure defines a rectangle that appears on the form.

Prototype

```
typedef struct {
  FormObjAttrType attr;
  RectangleType rect;
} FormRectangleType;
```

Your code should treat the `FormRectangleType` structure as opaque. Do not attempt to change structure member values directly.

Fields

`attr`
: See `FormObjAttrType`.

`rect`
: Location and size of the rectangle.

FormTitleType Struct

Purpose The `FormTitleType` structure defines the title of the form.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `FormTitleType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  RectangleType rect;
  char *text;
} FormTitleType;
```

Your code should treat the `FormTitleType` structure as opaque. Do not attempt to change structure member values directly.

Fields

`rect`
: The location and size of the title area.

`text`
: Text of the title.

FormType Struct

Purpose The `FormType` structure and supporting structures are defined below.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `FormType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  WindowType window;
  UInt16 formId;
  FormAttrType attr;
  WinHandle bitsBehindForm;
  FormEventHandlerType *handler;
  UInt16 focus;
  UInt16 defaultButton;
```

```
  UInt16 helpRscId;
  UInt16 menuRscId;
  UInt16 numObjects;
  FormObjListType *objects;
} FormType;
```

Your code should treat the `FormType` structure as opaque. Do not attempt to change structure member values directly.

Fields

`window`
: Structure of the window object that corresponds to the form. See `WindowType`. Access this field with `FrmGetWindowHandle()`.

`formId`
: ID number of the form, specified by the application developer. This ID value is part of the event data of `frmOpenEvent`. The ID should match the form's resource ID. Access this field with `FrmGetFormId()`.

`attr`
: Form object attributes. See `FormAttrType`.

`bitsBehindForm`
: Used to save all the bits behind the form so the screen can be properly refreshed when the form is closed. This field is for internal use only by modal forms.

`handler`
: Routine called when the form needs to handle an event. You typically set this in your application's event handling function by calling `FrmSetEventHandler()`.

`focus`
: Index of a field or table object within the form that contains the focus. Any `keyDownEvent` is passed to the object that has the focus. Set to `noFocus` if no object has the focus. Set this field with `FrmSetFocus()`.

`defaultButton`
: Resource ID of the object defined as the default button. This value is used by the routine `FrmDoDialog()`.

`helpRscId`
: Resource ID number of the help resource. The help resource is a String resource (type tSTR).

`menuRscId`
: ID number of a menu bar to use if the form has a menu, or zero if the form doesn't have a menu.

`numObjects`
: Number of objects contained within the form. Access this field with `FrmGetNumberOfObjects()`.

`objects`
: Pointer to the array of objects contained within the form. See `FormObjListType`.

FrmGraffitiStateType Struct

Purpose The `FrmGraffitiStateType` structure defines the shift indicator.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `FrmGraffitiStateType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct{
  PointerType pos;
}FrmGraffitiStateType;
```

Your code should treat the `FrmGraffitiStateType` structure as opaque. Do not attempt to change structure member values directly.

Fields

`pos`
: Location of the shift indicator.

FrmNavHeaderType Struct

Purpose Contains header information for the form's navigation order. Obtain this structure for a given form by calling `FrmGetNavOrder()`; set it by calling `FrmSetNavOrder()`.

Declared In `Form.h`

Prototype

```
typedef struct FrmNavHeaderTag {
   UInt16 version;
   UInt16 numberOfObjects;
   UInt16 headerSizeInBytes;
   UInt16 listElementSizeInBytes;
   FrmNavHeaderFlagsType navFlags;
   UInt16 initialObjectIDHint;
   UInt16 jumpToObjectIDHint;
   UInt16 bottomLeftObjectIDHint;
   UInt16 padding1;
} FrmNavHeaderType
```

Fields

`version`
: The version of this structure used by the device. The version of the structure defined in `Form.h` is specified by `kFrmNavInfoVersion`.

`numberOfObjects`
: The number of objects in the navigation order for this form.

`headerSizeInBytes`
: The size of this header structure, in bytes. The version 1 structure is 20 bytes in length.

`listElementSizeInBytes`
: Size of the `FrmNavOrderEntryType` structures that are used to represent each object in the navigation order.

`navFlags`
: A combination of the flags listed under `FrmNavHeaderFlagsType`. These flags specify the form's initial focus mode.

`initialObjectIDHint`
: ID of the object where focus is positioned when the form is initialized. If `initialObjectIDHint` is 0, the operating system places the initial focus on the first action button, if there is one, or on the first object in the tab order if there is not.

`jumpToObjectIDHint`
: ID of the object to which focus can "jump," if the device supports this feature. Devices can optionally have an action trigger the movement of the focus to a commonly used object.

`bottomLeftObjectIDHint`
: For platforms that cycle vertically, this field specifies which object receives the focus when navigating up from an object in the top row of the form (an object whose `aboveObjectID` [see `FrmNavOrderEntryType`] is 0). A `bottomLeftObjectIDHint` value of zero means that focus does not cycle vertically in the form.

`padding1`
: Structure padding bytes.

Compatibility Implemented only if One-Handed Navigation Feature Set is present.

FrmNavOrderEntryType Struct

Purpose Defines how navigation up or down using keys for that purpose should proceed from a given UI object.

Declared In `Form.h`

Prototype

```
typedef struct FrmNavOrderEntryTag {
   UInt16 objectID;
   FrmNavObjectFlagsType objectFlags;
   UInt16 aboveObjectID;
   UInt16 belowObjectID;
} FrmNavOrderEntryType
```

Fields

`objectID`
: ID of the object in the navigation order.

`objectFlags`
: One or more of the flags defined under `FrmNavObjectFlagsType`.

`aboveObjectID`
: ID of the object that is above the object being described. Should be 0 if the object being described is in the top row of the form. If the user navigates up from an object with 0 for its above object, some platforms will move the focus to the

object specified by `bottomLeftObjectIDHint` in the form's `FrmNavHeaderType` structure.

`belowObjectID`
ID of the object that is below the object being described. Should be 0 if the object is in the bottom row of the form. If the user navigates down from an object with 0 for its below object, some platforms will move the focus to the first object in the tab order.

Comments You can obtain an array of structures of this type for a given form by calling `FrmGetNavOrder()`. When setting the navigation order for a form, you create an array of these structures—one for each UI object in the navigation order—and then pass the address of that array to `FrmSetNavOrder()`.

NOTE: `FrmSetNavOrder()` is not intended to replace the use of navigation resources. A form that needs to have a custom navigation order should *always* provide a navigation resource. `FrmSetNavOrder()` is mainly for dynamically created forms or forms that have a navigation resource but change the order sometime after form initialization.

Compatibility Implemented only if One-Handed Navigation Feature Set is present.

Form Constants

AlertType Enum

Purpose Constants the define the types of alerts that can be presented to the user.

Declared In `Form.h`

Prototype

```
typedef enum alertTypes AlertType;
```

Constants `informationAlert`
An alert that simply informs the user of a non-critical situation.

`confirmationAlert`
An alert requiring that the user confirms the displayed information.

`warningAlert`
An alert that is informing the user of a potentially serious condition.

`errorAlert`
An alert that is informing the user that an error has occurred.

FormObjectKind Enum

Purpose The `FormObjectKind` enum specifies values for the `objectType` field of the `FormObjListType`. It specifies how to interpret the `object` field.

Declared In `Form.h`

Prototype

```
typedef enum formObjects FormObjectKind;
```

Constants

`frmFieldObj`
Text field.

`frmControlObj`
Control.

`frmListObj`
List.

`frmTableObj`
Table.

`frmBitmapObj`
Form bitmap.

`frmLineObj`
Line.

`frmFrameObj`
Frame.

`frmRectangleObj`
Rectangle.

`frmLabelObj`
Label.

`frmTitleObj`
Form title.

`frmPopupObj`
Popup list.

`frmGraffitiStateObj`
Graffiti® or Graffiti 2 shift indicator.

`frmGadgetObj`
Gadget (custom object).

`frmScrollBarObj`
Scrollbar.

FrmNavFocusRingStyleEnum Typedef

Purpose Constants that indicate the focus ring style.

Declared In `Form.h`

Prototype

```
typedef UInt16 FrmNavFocusRingStyleEnum
```

Constants

`frmNavFocusRingStyleObjectTypeDefault = 0`
Use a focus ring that matches the underlying object.

`frmNavFocusRingStyleSquare = 1`
A rectangular focus ring with square corners.

`frmNavFocusRingStyleRound = 2`
A rectangular focus ring with rounded corners.

`frmNavFocusRingStyleHorizontalBars = 3`
The focus ring simply consists only of horizontal lines.

`frmNavFocusRingStyleInvalid = 0xFFFF`
A value returned by `FrmNavGetFocusRingInfo()` to indicate that the ring style is invalid.

Comments Set a form object's ring style with `FrmNavDrawFocusRing()`. You can determine an existing object's ring style by calling `FrmNavGetFocusRingInfo()`.

If a chosen focus ring style is not supported by the device, `FrmNavDrawFocusRing()` will instead use `frmNavFocusRingStyleObjectTypeDefault`. Note that `frmNavFocusRingStyleHorizontalBars` is not supported on all devices.

Compatibility Implemented only if One-Handed Navigation Feature Set is present.

FrmNavHeaderFlagsType Typedef

Purpose Form navigation flags that primarily specify the form's initial focus mode (application focus mode or object focus mode). These flags make up the `navFlags` field of the `FrmNavHeaderType` structure.

Declared In `Form.h`

Prototype

```
typedef UInt32 FrmNavHeaderFlagsType
```

Constants

```
#define kFrmNavHeaderFlagsAppFocusStartState
  0x00000002
```

If this flag is set, the form will initially be in application focus mode.

```
#define kFrmNavHeaderFlagsAppFocusStartStateValue
  0x00000002
```

After masking the flag bits with `kFrmNavHeaderFlagsStartStateMask`, the result can be compared with this value to determine if the form will initially be in application focus mode.

```
#define kFrmNavHeaderFlagsAutoGenerated 0x80000000
```

```
#define kFrmNavHeaderFlagsDefaultStartStateValue
  0x00000000
```

After masking the flag bits with `kFrmNavHeaderFlagsStartStateMask`, the result can be compared with this value to determine if the form's initial focus mode will be chosen by the system.

```
#define kFrmNavHeaderFlagsInvalidStartStateValue
  0x00000003
```

After masking the flag bits with `kFrmNavHeaderFlagsStartStateMask`, the result can be compared with this value to determine if the focus mode flags are improperly set.

`#define kFrmNavHeaderFlagsObjectFocusStartState 0x00000001`

If this flag is set, the form will initially be in object focus mode.

`#define kFrmNavHeaderFlagsObjectFocusStartStateValue 0x00000001`

After masking the flag bits with `kFrmNavHeaderFlagsStartStateMask`, the result can be compared with this value to determine if the form will initially be in object focus mode.

`#define kFrmNavHeaderFlagsStartStateMask 0x00000003`

A mask that allows you to isolate the "start state" bits and treat the result as a value. The result can be compared with any of the `kFrmNavHeaderFlags...StartStateValue` constants to determine the starting state.

Comments

If neither `kFrmNavHeaderFlagsObjectFocusStartState` nor `kFrmNavHeaderFlagsAppFocusStartState` is set, then the system decides whether to initially put the form in object focus mode or application focus mode. It is an error if both flags are set.

Example

The following code excerpt shows how you can check if a form's initial state will be decided by the system.

```
if ( (navFlags & kFrmNavHeaderFlagsStartStateMask) ==
kFrmNavHeaderFlagsDefaultStartStateValue)) {
   // Form's initial state is decided by system
}
```

Compatibility

Implemented only if One-Handed Navigation Feature Set is present.

FrmNavObjectFlagsType Typedef

Purpose Flags that specify different aspects of a navigation object.

Declared In `Form.h`

Prototype

```
typedef UInt16 FrmNavObjectFlagsType
```

Constants

`#define kFrmNavObjectFlagsForceInteractionMode 0x0002`

Used with multi-line text fields, if this flag is set the field is put in interaction mode when it receives the focus. Otherwise, the field is drawn in the focused, non-interaction mode state when it receives the focus.

`#define kFrmNavObjectFlagsIsBigButton 0x8000`

The object is a "big button." Big buttons are a type of UI object created by palmOne. Basically, they are thumbable buttons with a bitmap and some text. They are implemented as gadgets.

`#define kFrmNavObjectFlagsSkip 0x0001`

If this flag is set, the object is skipped when focus moves from object to object. The object is included in the order only when it explicitly gets focus (which primarily occurs when a field, table with a field, popup trigger, or selector trigger gets focus by being tapped with the pen).

Comments The flags for a given control in the navigation order can be obtained by calling `FrmGetNavEntry()`; you can set them for a control by calling `FrmSetNavEntry()`. In both cases the flags are one of the fields of a `FrmNavOrderEntryType` structure.

Compatibility Implemented only if One-Handed Navigation Feature Set is present.

FrmNavStateFlagsType Typedef

Purpose Flags that define a form's navigation state. Use `FrmGetNavState()` to retrieve a form's navigation state. Use `FrmSetNavState()` to set it.

Declared In `Form.h`

Prototype

```
typedef UInt32 FrmNavStateFlagsType
```

Constants

```
#define kFrmNavStateFlagsInteractionMode
  0x00000001
```

The form is in "interaction mode." If this bit is not set, the form is in "navigation mode." See the Comments section, below, for information on what these modes mean.

```
#define kFrmNavStateFlagsObjectFocusMode
  0x00000002
```

The form is in "object focus mode." If this bit is not set, the form is in "application focus mode." See the Comments section, below, for information on what these modes mean.

Comments

Interaction Mode vs. Navigation Mode

More complex objects need to "take over" the scroll keys in order to interact with them. For example, a text field needs to allocate all four scroll keys to move the cursor. Similarly, a pop-up list needs to use up and down to change the selection in the list. This conflicts with the requirement of using the scroll keys to navigate between objects. As a result, these complex objects need to have an **interaction mode**, where they can take over control of the keys. The opposite of interaction mode is **navigation mode**, where scroll keys navigate between objects. On a system with one-handed navigation, pressing "center" toggles between interaction mode and navigation mode. Finally, a subset of interaction mode is edit mode, which refers specifically to text fields.

Object Focus Mode vs. Application Focus Mode

Interaction mode and Navigation mode are palmOne concepts. PalmSource has defined application focus mode and object focus mode. **Application focus mode** refers to applications that do not have keyboard navigation enabled. In this state, up and down act as page up and page down in the traditional method that Palm OS implemented in its original form. **Object focus mode** refers to the

state where individual objects on the screen can receive focus, essentially what "navigation mode" refers to above.

Applications may or may not be able to toggle between application focus and object focus modes.

Compatibility Implemented only if One-Handed Navigation Feature Set is present.

Form Navigation Focus Error Codes

Purpose Error codes returned by the `FrmNav...` and `Frm...Nav...` functions.

Declared In `Form.h`

Constants

```
#define uilibErrCurrentFocusInvalid
  (uilibErrorClass | 2)
```
There is no current focus object.

```
#define uilibErrFormDoesNotHaveFocus
  (uilibErrorClass | 7)
```
The specified form doesn't currently have the focus.

```
#define uilibErrInvalidFocusObject
  (uilibErrorClass | 6)
```
The control to which focus is being transferred is not marked as usable.

```
#define uilibErrInvalidParam (uilibErrorClass | 1)
```
At least one of the supplied parameters is invalid.

```
#define uilibErrNoNavInfoForForm (uilibErrorClass | 5)
```
The specified form has no navigation information.

```
#define uilibErrObjectFocusModeOff
  (uilibErrorClass | 3)
```
The specified form is not in object focus mode. For the `FrmGlueNav...` functions in the Palm OS Glue library, this error code is also used to indicate that the device doesn't support one-handed navigation.

```
#define uilibErrObjectNotFound (uilibErrorClass | 4)
```
There is no object with a system-drawn focus ring.

Compatibility Implemented only if One-Handed Navigation Feature Set is present.

Form Dynamic Input Area Policies

Purpose A dynamic input area policy specifies how the dynamic input area should be handled while a form is active. These values are used for the `diaPolicy` attribute in a form's attribute structure. You can set the value with `FrmSetDIAPolicyAttr` and retrieve it with `FrmGetDIAPolicyAttr`.

Declared In `Form.h`

Constants

`#define frmDIAPolicyCustom 1`
: The user and the application control whether the input area is active.

`#define frmDIAPolicyStayOpen 0`
: Forces the dynamic input area to stay open while the form is active. The input trigger is disabled.

Compatibility Implemented only if Pen Input Manager Feature Set is present.

Miscellaneous Form Constants

Purpose These form constants are also defined.

Declared In `Form.h`

Constants

`#define formNavRscType 'fnav'`
: (defined in `UIResources.h`) The resource type of a navigation resource. It is defined as a 68K format (big endian) resource.

`#define frmInvalidObjectId 0xffff`
: A constant value that represents an invalid object ID or index.

`#define frmNavFocusRingNoExtraInfo ((Int16)0xFFFF)`
: Value that indicates that there is no extra information associated with current navigation object. Pass this value to `FrmNavDrawFocusRing()` if you have no such information to associate with the object getting the focus ring. This value is returned from `FrmNavGetFocusRingInfo()` if there

was no extra information associated with the object around which the focus ring is currently drawn.

`#define frmNoSelectedControl 0xff`
Returned by `FrmGetControlGroupSelection()` if no control is selected.

`#define frmRedrawUpdateCode 0x8000`
Indicates that the form should be redrawn; flag in a `frmUpdateEvent`.

`#define frmResponseCreate 1974`
Passed to `FormCheckResponseFuncType()` to indicate that the function should perform initialization.

`#define frmResponseQuit ((Int16) 0xBEEF)`
Passed to `FormCheckResponseFuncType()` to indicate that the function should perform cleanup.

`#define kFrmNavInfoVersion 1`
The version of the `FrmNavHeaderType` and `FrmNavOrderEntryType` structures declared in `Form.h`.

`#define noFocus 0xffff`
No form object has the focus.

One-Handed Navigation Trap Selectors

Purpose Trap selectors for the one-handed navigation functions.

Declared In `Form.h`

Constants

```
#define NavSelectorFrmCountObjectsInNavOrder 0x0
#define NavSelectorFrmGetNavEntry 0x3
#define NavSelectorFrmGetNavOrder 0x1
#define NavSelectorFrmGetNavState 0x5
#define NavSelectorFrmNavDrawFocusRing 0x7
#define NavSelectorFrmNavGetFocusRingInfo 0x9
#define NavSelectorFrmNavObjectTakeFocus 0xA
#define NavSelectorFrmNavRemoveFocusRing 0x8
#define NavSelectorFrmSetNavEntry 0x4
```

```
#define NavSelectorFrmSetNavOrder 0x2
#define NavSelectorFrmSetNavState 0x6
```

Compatibility Implemented only if One-Handed Navigation Feature Set is present.

Form Resources

The following resources are associated with forms and with the objects on a form whose data structures are defined above:

- Form—Form Resource (tFRM)
- Alert dialog— Alert Resource (Talt)
- Bitmap—Form Bitmap Resource (tFBM)
- Button—Button Resource (tBTN)
- Check box—Check Box Resource (tCBX)
- Field—Field Resource (tFLD)
- Gadget (custom object)—Gadget Resource (tGDT)
- Shift indicator—Shift Indicator Resource (tGSI)
- Label—Label Resource (tLBL)
- List—List Resource (tLST)
- Popup trigger—Popup Trigger Resource (tPUT)
- Push button—Push Button Resource (tPBN)
- Repeating button—Repeating Button Resource (tREP)
- Scrollbar—Scroll Bar Resource (tSCL)
- Selector trigger—Selector Trigger Resource (tSLT)
- Table—Table Resource (tTBL)

See *Palm OS User Interface Guidelines* for more information on these and other resources.

Form Functions

ECFrmValidatePtr Macro

Returns `true` if `ERROR_CHECK_LEVEL` is equal to `ERROR_CHECK_FULL` and if the specified pointer references a valid form.

Declared In `Form.h`

Prototype

```
#define ECFrmValidatePtr (formP)
```

Parameters

→ *formP*
Pointer to be tested.

Returns Returns `true` if `ERROR_CHECK_LEVEL` is equal to `ERROR_CHECK_FULL` and if the specified pointer is a non-`NULL` pointer to an object having a valid form structure. If `ERROR_CHECK_LEVEL` is not equal to `ERROR_CHECK_FULL`, this macro does nothing.

Comments This macro simply calls `FrmValidatePtr()` function if `ERROR_CHECK_LEVEL` is equal to `ERROR_CHECK_FULL`, and does nothing if it is not. Since `FrmValidatePtr()` is intended for debugging purposes only and should not be included in released code, this macro can be used to conditionally compile calls to `FrmValidatePtr()`.

Compatibility `FrmValidatePtr()` is implemented only if 3.0 New Feature Set is present.

FrmAlert Function

Purpose Create a modal dialog from an alert resource and display it until the user selects a button in the dialog.

Declared In `Form.h`

Prototype

```
UInt16 FrmAlert (UInt16 alertId)
```

Parameters

→ *alertId*
ID of the alert resource.

Returns Returns the item number of the button the user selected. A button's item number is determined by its order in the alert dialog; the first button has the item number 0 (zero).

NOTE: A default button press is simulated if the user switches to a different application while a modal dialog is active.

See Also `FrmDoDialog()`, `FrmCustomAlert()`, `FrmCustomResponseAlert()`

FrmCloseAllForms Function

Purpose Send a `frmCloseEvent` to all open forms.

Declared In `Form.h`

Prototype

```
void FrmCloseAllForms (void)
```

Parameters None.

Returns Returns nothing.

Comments Applications can call this function to ensure that all forms are closed cleanly before exiting `PilotMain()`; that is, before termination.

See Also `FrmSaveAllForms()`

FrmCopyLabel Function

Purpose Copy the passed string into the data structure of the specified label object in the active form.

Declared In `Form.h`

Prototype

```
void FrmCopyLabel (FormType *formP,
   UInt16 labelID, const Char *newLabel)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *labelID*
ID of form label object.

→ *newLabel*
Pointer to a null-terminated string.

Returns Returns nothing.

Comments The size of the new label *must not* exceed the size of the label defined in the resource. When defining the label in the resource, specify an initial size at least as big as any of the strings that will be assigned dynamically. This function redraws the label if the form's `usable` attribute and the label's `visible` attribute are set.

This function redraws the label but does not erase the old one first. If the new label is shorter than the old one, the end of the old label will still be visible. To avoid this, you can hide the label using `FrmHideObject()`, then show it using `FrmShowObject()`, after using `FrmCopyLabel`.

Note that `FrmCopyLabel` copies the passed string into memory already allocated for the label. Thus, the string doesn't need to remain in existence once `FrmCopyLabel` returns.

See Also `FrmGetLabel()`

FrmCopyTitle Function

Purpose Copy a new title over the form's current title. If the form is visible, the new title is drawn.

Declared In `Form.h`

Prototype

```
void FrmCopyTitle (FormType *formP,
    const Char *newTitle)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *newTitle*
Pointer to the new title string.

Returns Returns nothing.

Comments The size of the new title **must not** exceed the title size defined in the resource. When defining the title in the resource, specify an initial size at least as big as any of the strings to be assigned dynamically.

See Also `FrmGetTitle()`, `FrmSetTitle()`

FrmCountObjectsInNavOrder Function

Purpose Obtains the number of objects in a form's navigation tab order.

Declared In `Form.h`

Prototype

```
UInt16 FrmCountObjectsInNavOrder
    (const FormType *formP)
```

Parameters → *formP*
Pointer to the form whose navigation information is to be obtained.

Returns The number of objects in the navigation tab order.

Comments If a form does not have a navigation resource, all objects in the form are included in the tab order. Note, however, that many will be marked as skipped.

Compatibility Implemented only if One-Handed Navigation Feature Set is present.

See Also `FrmGetNavOrder()`

FrmCustomAlert Function

Purpose Create a modal dialog from an alert resource and display the dialog until the user taps a button in the alert dialog.

Declared In `Form.h`

Prototype

```
UInt16 FrmCustomAlert (UInt16 alertId,
    const Char *s1, const Char *s2,
    const Char *s3)
```

Parameters → *alertId*
Resource ID of the alert.

→ *s1, s2, s3*
Strings to replace ^1, ^2, and ^3 (see Comments).

Returns Returns the number of the button the user tapped (the first button is zero).

Comments A button's item number is determined by its order in the alert template; the first button has the item number zero.

Up to three strings can be passed to this routine. They are used to replace the variables ^1, ^2 and ^3 that are contained in the message string of the alert resource.

If the variables ^1, ^2, and ^3 occur in the message string, do not pass `NULL` for the arguments `s1`, `s2`, and `s3`. If you want an argument to be ignored, pass the empty string (`""`). In Palm OS 2.0 or below, pass a string containing a space (`" "`) instead of the empty string.

NOTE: A default button press is simulated if the user switches to a different application while a modal dialog is active.

See Also `FrmAlert()`, `FrmDoDialog()`, `FrmCustomResponseAlert()`

FrmCustomResponseAlert Function

Purpose Create a modal dialog with a text field from an alert resource and display it until the user taps a button in the alert dialog.

Declared In `Form.h`

Prototype

```
UInt16 FrmCustomResponseAlert (UInt16 alertId,
    const Char *s1, const Char *s2,
    const Char *s3,
    Char *entryStringBuf,
    Int16 entryStringBufLength,
    FormCheckResponseFuncPtr callback)
```

Parameters

→ *alertId*
Resource ID of the alert.

→ *s1, s2, s3*
Strings to replace ^1, ^2, and ^3. See the Comments in `FrmCustomAlert()` for more information.

← *entryStringBuf*
The string the user entered in the text field.

→ *entryStringBufLength*
The maximum length for the string in `entryStringBuf`.

→ *callback*
A callback function that processes the string. See `FormCheckResponseFuncType()`. Pass `NULL` if there is no callback.

Returns Returns the number of the button the user tapped (the first button is zero).

Comments This function differs from `FrmCustomAlert()` in these ways:

- The dialog it displays contains a text field for user entry. The text that the user enters is returned in the `entryStringBuf` parameter.
- When the user taps a button, the `callback` function is called and is passed the button number and `entryStringBuf`. The dialog is only dismissed if the callback returns `true`. This behavior allows you to perform error checking on the string that the user entered and give the user a chance to re-enter the string.

  The callback function is also called with special constants when the alert dialog is being initialized and when it is being deallocated. This allows the callback to perform any necessary initialization and cleanup.

NOTE: A default button press is simulated if the user switches to a different application while a modal dialog is active.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `FrmAlert()`, `FrmDoDialog()`

FrmDeleteForm Function

Purpose Release the memory occupied by a form. Any memory allocated to objects in the form is also released.

Declared In `Form.h`

Prototype
```
void FrmDeleteForm (FormType *formP)
```

Parameters → *formP*
Pointer to the form object (`FormType` structure).

Returns Returns nothing.

Comments This function doesn't modify the display.

Compatibility If 3.5 New Feature Set is present and the form contains an extended gadget, this function calls the gadget's callback with `formGadgetDeleteCmd`. See `FormGadgetHandlerType()`.

See Also `FrmInitForm()`, `FrmReturnToForm()`

FrmDispatchEvent Function

Purpose Dispatch an event to the application's handler for the form.

Declared In `Form.h`

Prototype

```
Boolean FrmDispatchEvent (EventType *eventP)
```

Parameters → *eventP*
Pointer to an event.

Returns Returns the Boolean value returned by the form's event handler or `FrmHandleEvent()`. (If the form's event handler returns `false`, the event is passed to `FrmHandleEvent`.) This function also returns false if the form specified in the event is invalid.

Comments The event is dispatched to the current form's handler unless the form ID is specified in the event data, as, for example, with `frmOpenEvent` or `frmGotoEvent`. A form's event handler (`FormEventHandlerType()`) is registered by `FrmSetEventHandler()`.

Note that if the form does not have a registered event handler, this function causes a fatal error.

FrmDoDialog Function

Purpose Display a modal dialog until the user taps a button in the dialog.

Declared In `Form.h`

Prototype

```
UInt16 FrmDoDialog (FormType *formP)
```

Parameters → *formP*
Pointer to the form object (`FormType` structure).

Returns Returns the resource ID of the button the user tapped.

NOTE: A default button press is simulated if the user switches to a different application while a modal dialog is active.

Comments Before calling FrmDoDialog you must have called `FrmInitForm()` to load and initialize the dialog and you must have then set the event handler, if one is needed. After the call, read any values needed from the dialog's objects and then call `FrmDeleteForm()` to release the memory occupied by the dialog.

See Also `FrmInitForm()`, `FrmCustomAlert()`, `FrmCustomResponseAlert()`

FrmDrawForm Function

Purpose Draw all objects in a form and the frame around the form.

Declared In `Form.h`

Prototype

```
void FrmDrawForm (FormType *formP)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

Returns Returns nothing.

Comments If the `saveBehind` form attribute is set and the form is visible, this function saves the bits behind the form using the `bitsBehindForm` field in the `FormType` structure.

You should call this function in response to a `frmOpenEvent`.

If you do any custom drawing, you should do so after you call this function not before. If you do custom drawing, respond to `frmUpdateEvent` as well as `frmOpenEvent`, and be sure to return true to specify that the `frmUpdateEvent` was handled. The default event handler for `frmUpdateEvent` calls `FrmDrawForm`, so if you allow the event to fall through by returning `false`, your custom drawing is erased.

Compatibility If 3.5 New Feature Set is present, `FrmDrawForm` erases the form's window before performing any drawing. Thus, it is especially important to do any custom drawing after this function call on Palm OS 3.5 and higher.

If 3.5 New Feature Set is present and the form contains an extended gadget, this function calls the gadget's callback with `formGadgetDrawCmd`. See `FormGadgetHandlerType()`.

See Also `FrmEraseForm()`, `FrmInitForm()`

FrmEraseForm Function

Purpose Erase a form from the display.

Declared In `Form.h`

Prototype

```
void FrmEraseForm (FormType *formP)
```

Parameters → *formP*
Pointer to the form object (`FormType` structure).

Returns Returns nothing.

Comments If the region obscured by the form was saved by `FrmDrawForm()`, this function restores that region.

FrmGetActiveField Function

Purpose Return the active field for a specified form.

Declared In `Form.h`

Prototype

```
FieldType *FrmGetActiveField
    (const FormType *formP)
```

Parameters → *formP*
Pointer to the form for which the active field should be returned, or `NULL` if the active field on the active form is desired.

Returns Returns a pointer to the field object of the active field, or `NULL` if the form doesn't have an active field or if there is no active form.

Comments This function will most often be called with a `NULL` parameter to obtain the active field on the active form.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `FrmGetActiveForm()`

FrmGetActiveForm Function

Purpose Return the currently active form.

Declared In `Form.h`

Prototype
```
FormType *FrmGetActiveForm (void)
```

Parameters `None.`

Returns Returns a pointer to the form object of the active form.

Comments You should not call the `FrmGetActiveForm` function when a popup window is open. There is no active form while a popup is displayed, and the value returned from `FrmGetActiveForm` in this situation has no meaning.

See Also `FrmGetActiveField()`, `FrmGetActiveFormID()`, `FrmSetActiveForm()`

FrmGetActiveFormID Function

Purpose Return the ID of the currently active form.

Declared In `Form.h`

Prototype
```
UInt16 FrmGetActiveFormID (void)
```

Parameters `None.`

Returns Returns the active form's ID number.

See Also `FrmGetActiveForm()`

FrmGetControlGroupSelection Function

Purpose Return the item number of the control selected in a group of controls.

Declared In `Form.h`

Prototype
```
UInt16 FrmGetControlGroupSelection
    (const FormType *formP, UInt8 groupNum)
```

Parameters → *formP*
Pointer to the form object (`FormType` structure).

→ *groupNum*
Control group number.

Returns Returns the item number of the selected control; returns `frmNoSelectedControl` if no item is selected.

Comments The item number is the index into the form object's data structure.

NOTE: `FrmSetControlGroupSelection` sets the selection in a control group based on an object ID, *not* its index, which `FrmGetControlGroupSelection` returns.

Compatibility On versions prior to 3.5, this function returned a `Byte` instead of `UInt16`.

See Also `FrmGetObjectId()`, `FrmGetObjectPtr()`, `FrmSetControlGroupSelection()`

FrmGetControlValue Function

Purpose Return the current value of a control.

Declared In `Form.h`

Prototype

```
Int16 FrmGetControlValue (const FormType *formP,
    UInt16 objIndex)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *objIndex*
Index of the control object in the form object's data structure. You can obtain this by using `FrmGetObjectIndex()`.

Returns Returns the current value of the control. For most controls the return value is either 0 (off) or 1 (on). For sliders, this function returns the value of the `value` field.

Comments The caller must specify a valid index. This function is valid only for push button and check box control objects.

See Also `FrmSetControlValue()`

FrmGetDIAPolicyAttr Function

Purpose Returns a form's dynamic input area policy.

Declared In `Form.h`

Prototype
```
UInt16 FrmGetDIAPolicyAttr (FormPtr formP)
```

Parameters → *formP*
A pointer to a `FormType` structure.

Returns Returns one of the constants listed in "Form Dynamic Input Area Policies" on page 287.

Compatibility Implemented only if the Pen Input Manager Feature Set is present.

See Also `FrmSetDIAPolicyAttr`

FrmGetFirstForm Function

Purpose Return the first form in the window list.

Declared In `Form.h`

Prototype
```
FormType *FrmGetFirstForm (void)
```

Parameters `None.`

Returns Returns a pointer to a form object, or `NULL` if there are no forms.

Comments The window list is a LIFO stack. The last window created is the first window in the window list.

FrmGetFocus Function

Purpose Return the item (index) number of the object that has the focus.

Declared In `Form.h`

Prototype
```
UInt16 FrmGetFocus (const FormType *formP)
```

Parameters → *formP*
Pointer to the form object (`FormType` structure).

Returns Returns the index of the object (UI element) that has the focus, or returns `noFocus` if none does. To convert the object index to an ID, use `FrmGetObjectId()`.

See Also `FrmGetObjectPtr()`, `FrmSetFocus()`

FrmGetFormBounds Function

Purpose Return the visual bounds of the form; the region returned includes the form's frame.

Declared In `Form.h`

Prototype

```
void FrmGetFormBounds (const FormType *formP,
    RectangleType *rP)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

← *rP*
Pointer to a RectangleType structure where the bounds is returned.

Returns Returns nothing. The bounds of the form are returned in `r`.

FrmGetFormId Function

Purpose Return the resource ID of a form.

Declared In `Form.h`

Prototype

```
UInt16 FrmGetFormId (const FormType *formP)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

Returns Returns form resource ID.

See Also `FrmGetFormPtr()`

FrmGetFormPtr Function

Purpose Return a pointer to the form that has the specified ID.

Declared In `Form.h`

Prototype

```
FormType *FrmGetFormPtr (UInt16 formId)
```

Parameters

→ *formId*
Form ID number.

Returns Returns a pointer to the form object, or `NULL` if the form is not in memory.

See Also `FrmGetFormId()`

FrmGetGadgetData Function

Purpose Return the value stored in the `data` field of the gadget object.

Declared In `Form.h`

Prototype

```
void *FrmGetGadgetData (const FormType *formP,
    UInt16 objIndex)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *objIndex*
Index of the gadget object in the form object's data structure. You can obtain this by using `FrmGetObjectIndex()`.

Returns Returns a pointer to the custom gadget's data.

Comments Gadget objects provide a way for an application to attach custom gadgetry to a form. In general, the `data` field of a gadget object contains a pointer to the custom object's data structure.

See Also `FrmSetGadgetData()`, `FrmSetGadgetHandler()`

FrmGetLabel Function

Purpose Return pointer to the text of the specified label object in the specified form.

Declared In `Form.h`

Prototype

```
const Char *FrmGetLabel (const FormType *formP,
    UInt16 labelID)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *labelID*
ID of the label object.

Returns Returns a pointer to the label string.

Comments Does not make a copy of the string; returns a pointer to the string. The object must be a label.

See Also FrmCopyLabel()

FrmGetNavEntry Function

Purpose Gets the navigation information for a single object in the form.

Declared In Form.h

Prototype

```
Err FrmGetNavEntry (const FormType *formP,
    UInt16 targetObjectID, UInt16 *afterObjectIDP,
    UInt16 *aboveObjectIDP,
    UInt16 *belowObjectIDP,
    FrmNavObjectFlagsType *objectFlagsP)
```

Parameters

→ *formP*
Pointer to the form whose navigation order is being obtained.

→ *targetObjectID*
The ID of the "target object"—the object whose navigation information is being obtained.

← *afterObjectIDP*
Pointer to a UInt16 into which will be written the ID of the object which is to the left of the target object. If the target object is the first one in the tab order, a value of 0 will be written to *afterObjectIDP*. Pass NULL if you do not need this information.

← *aboveObjectIDP*
Pointer to a UInt16 into which will be written the ID of the object that is above the target object. If the target object is in the top row of the form, a value of 0 will be written to *aboveObjectIDP*. Pass NULL if you do not need this information.

← *belowObjectIDP*
Pointer to a UInt16 into which will be written the ID of the object that is below the target object. If the target object is in the bottom row of the form, a value of 0 will be written to *belowObjectIDP*. Pass NULL if you do not need this information.

← *objectFlagsP*
Pointer to a FrmNavObjectFlagsType into which will be written the object flags for the target object. Pass NULL if you do not need this information.

Returns errNone if the operation completed successfully, or an error code otherwise.

Compatibility Implemented only if One-Handed Navigation Feature Set is present.

See Also FrmGetNavOrder(), FrmSetNavEntry()

FrmGetNavOrder Function

Purpose Obtains information about the navigation order of a form.

Declared In Form.h

Prototype

```
Err FrmGetNavOrder (const FormType *formP,
    FrmNavHeaderType *navHeaderP,
    FrmNavOrderEntryType *navOrderP,
    UInt16 *numObjectsP)
```

Parameters

→ *formP*
Pointer to the form whose focus order is to be obtained.

← *navHeaderP*
If this pointer is not NULL, the caller-supplied FrmNavHeaderType structure is filled with the header information for the form's navigation order.

← *navOrderP*
If this pointer is not NULL, the caller-supplied array of FrmNavOrderEntryType structures is filled with information for each object in the navigation order.

↔ *numObjectsP*
On entry, the number of entries that you allocated in the navigation order array. On exit, this is set to the number of entries that were filled in in the navigation order array.

Returns errNone if the operation completed successfully, or an error code otherwise.

Comments If a form does not have a navigation resource, all objects in the form are included in the tab order. Note, however, that many will be marked as skipped.

Compatibility Implemented only if One-Handed Navigation Feature Set is present.

See Also `FrmCountObjectsInNavOrder()`, `FrmSetNavOrder()`

FrmGetNavState Function

Purpose Gets the current navigation state of a form.

Declared In `Form.h`

Prototype

```
Err FrmGetNavState (const FormType *formP,
    FrmNavStateFlagsType *stateFlagsP)
```

Parameters

→ *formP*
Pointer to the form whose navigation state is to be obtained.

← *stateFlagsP*
Pointer to a `FrmNavStateFlagsType` into which the current navigation state is written. This state identifies the form's current focus mode.

Returns `errNone` if the operation completed successfully, or an error code otherwise.

Compatibility Implemented only if One-Handed Navigation Feature Set is present.

See Also `FrmSetNavState()`

FrmGetNumberOfObjects Function

Purpose Return the number of objects in a form.

Declared In `Form.h`

Prototype

```
UInt16 FrmGetNumberOfObjects
    (const FormType *formP)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

Returns Returns the number of objects in the specified form.

See Also FrmGetObjectPtr(), FrmGetObjectId()

FrmGetObjectBounds Function

Purpose Retrieve the bounds of an object given its form and index.

Declared In Form.h

Prototype

```
void FrmGetObjectBounds (const FormType *formP,
    UInt16 objIndex, RectangleType *rP)
```

Parameters

→ *formP*
Pointer to the form object (FormType structure).

→ *objIndex*
Index of an object in the form. You can obtain this by using FrmGetObjectIndex().

← *rP*
Pointer to a RectangleType structure where the object bounds are returned. The bounds are in window-relative coordinates.

Returns Returns nothing. The object's bounds are returned in r.

See Also FrmGetObjectPosition(), FrmSetObjectBounds(), FrmSetObjectPosition()

FrmGetObjectId Function

Purpose Return the ID of the specified object.

Declared In Form.h

Prototype

```
UInt16 FrmGetObjectId (const FormType *formP,
    UInt16 objIndex)
```

Parameters

→ *formP*
Pointer to the form object (FormType structure).

→ *objIndex*
Index of an object in the form. You can obtain this by using FrmGetObjectIndex().

Returns Returns the ID number of an object or `frmInvalidObjectId` if the `objIndex` parameter is invalid.

See Also `FrmGetObjectPtr()`

FrmGetObjectIndex Function

Purpose Return the index of an object in the form's objects list.

Declared In `Form.h`

Prototype

```
UInt16 FrmGetObjectIndex (const FormType *formP,
    UInt16 objID)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *objID*
ID of an object in the form.

Returns Returns the index of the specified object (the index of the first object is 0), or `frmInvalidObjectId` if the supplied object ID is invalid.

Comments Bitmaps use a different mechanism for IDs than the rest of the form objects. When finding a bitmap with `FrmGetObjectIndex`, you need to pass the bitmap's resource ID, not the ID of the form bitmap object. (Passing the ID of the form bitmap object may or may not give you the right object back, depending on how you created the objects.)

This means that if you've got the same bitmap in two different form bitmap objects on the same form, you won't be able to use `FrmGetObjectIndex` to get at the second one; it'll always return the first.

See Also `FrmGetObjectPtr()`, `FrmGetObjectId()`

FrmGetObjectIndexFromPtr Function

Purpose Return an object's index.

Declared In `Form.h`

Prototype

```
UInt16 FrmGetObjectIndexFromPtr
    (const FormType *formP, void *objP)
```

Parameters

→ *formP*
Pointer to a `FormType`.

→ *objP*
Pointer to an object.

Returns Returns the object's index. `frmInvalidObjectId` is returned if `objP` is not associated with the form.

Comments Many form functions (`FrmGetObjectType()`, `FrmHideObject()`, and `FrmGetObjectBounds()`, for example) take an object index as one of their arguments. The most common way to get an object's index is to call `FrmGetObjectIndex()`. `FrmGetObjectIndex` takes a form ID and returns the form object's index. This is the routine one should use in most cases, because the application usually knows the object ID. However, gadgets and specifically extended gadgets, have APIs with callbacks that pass back the gadget pointer and not the ID. In those cases, the only way to get the object index (so one can use the `FrmGetObject*` APIs) is to use `FrmGetObjectIndexFromPtr`.

If you need the same functionality on pre-Palm OS 4.0 systems then you can accomplish the same thing with the following code snippet.

```
UInt16 index;
UInt16 objIndex = frmInvalidObjectId;
UInt16 numObjects = FrmGetNumberOfObjects(frmP)
for (index = 0; index < numObjects; index++) {
     if (FrmGetObjectPtr(index) == myObjPtr) {
          // Found it
          objIndex = index;
          break;
     }
}
```

Compatibility Implemented only if 4.0 New Feature Set is present.

FrmGetObjectPosition Function

Purpose Return the coordinates of the specified object relative to the form.

Declared In `Form.h`

Prototype

```
void FrmGetObjectPosition (const FormType *formP,
    UInt16 objIndex, Coord *x, Coord *y)
```

Parameters

→ *formP*
Pointer to the form object (FormType structure).

→ *objIndex*
Index of an object in the form. You can obtain this by using FrmGetObjectIndex().

← *x, y*
Pointers where the window-relative x and y positions of the object are returned. These locate the top-left corner of the object.

Returns Returns nothing.

See Also FrmGetObjectBounds(), FrmSetObjectPosition()

FrmGetObjectPtr Function

Purpose Return a pointer to the data structure of an object in a form.

Declared In `Form.h`

Prototype

```
void *FrmGetObjectPtr (const FormType *formP,
    UInt16 objIndex)
```

Parameters

→ *formP*
Pointer to the form object (FormType structure).

→ *objIndex*
Index of an object in the form. You can obtain this by using FrmGetObjectIndex().

Returns Returns a pointer to an object in the form.

See Also FrmGetObjectId()

FrmGetObjectType Function

Purpose Return the type of an object.

Declared In `Form.h`

Prototype

```
FormObjectKind FrmGetObjectType
    (const FormType *formP, UInt16 objIndex)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *objIndex*
Index of an object in the form. You can obtain this by using `FrmGetObjectIndex()`.

Returns Returns the type of the specified object. See `FormObjectKind` for the set of values that can be returned by this function.

FrmGetTitle Function

Purpose Return a pointer to the title string of a form.

Declared In `Form.h`

Prototype

```
const Char *FrmGetTitle (const FormType *formP)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

Returns Returns a pointer to title string, or `NULL` if there is no title string or there is an error finding it.

Comments This is a pointer to the internal structure itself, *not* to a copy.

See Also `FrmCopyTitle()`, `FrmSetTitle()`

FrmGetWindowHandle Function

Purpose Return the window handle of a form.

Declared In `Form.h`

Prototype

```
WinHandle FrmGetWindowHandle
    (const FormType *formP)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

Returns Returns the handle of the memory block that contains the form data structure. Since the form structure begins with the `WindowType`, this is also a `WinHandle`.

FrmGotoForm Function

Purpose Send a `frmCloseEvent` to the current form; send a `frmLoadEvent` and a `frmOpenEvent` to the specified form.

Declared In `Form.h`

Prototype

```
void FrmGotoForm (UInt16 formId)
```

Parameters → *formId*
ID of the form to display.

Returns Returns nothing.

Comments The default form event handler (`FrmHandleEvent()`) erases and disposes of a form when it receives a `frmCloseEvent`.

See Also `FrmPopupForm()`

FrmHandleEvent Function

Purpose Handle the event that has occurred in the form.

Declared In `Form.h`

Prototype

```
Boolean FrmHandleEvent (FormType *formP,
   EventType *eventP)
```

Parameters → *formP*
Pointer to the form object (`FormType` structure).

→ *eventP*
Pointer to the event data structure (`EventType`).

Returns Returns `true` if the event was handled.

Comments Never call this function directly. Call `FrmDispatchEvent()` instead. `FrmDispatchEvent` passes events to a form's custom event handler and then, if the event was not handled, to this function.

WARNING! You should never call this function directly. You should call the FrmDispatchEvent function instead.

Table 11.1 provides an overview of how `FrmHandleEvent` handles different events.

Table 11.1 FrmHandleEvent Actions

When FrmHandleEvent receives...	FrmHandleEvent performs these actions...
`ctlEnterEvent`	Passes the event and a pointer to the object the event occurred in to `CtlHandleEvent()`. The object pointer is obtained from the event data. If the control is part of an exclusive control group, it deselects the currently selected control of the group first.
`ctlRepeatEvent`	Passes the event and a pointer to the object the event occurred in to `CtlHandleEvent`. The object pointer is obtained from the event data.
`ctlSelectEvent`	Checks if the control is a Popup Trigger Control. If it is, the list associated with the popup trigger is displayed until the user makes a selection or touches the pen outside the bounds of the list. If a selection is made, a `popSelectEvent` is added to the event queue.
`fldEnterEvent` or `fldHeightChangedEvent`	Checks if a field object or a table object has the focus and passes the event to the appropriate handler (`FldHandleEvent()` or `TblHandleEvent()`). The table object is also a container object, which may contain a field object. If `TblHandleEvent` receives a field event, it passes the event to the field object contained within it.
`frmCloseEvent`	Erases the form and releases any memory allocated for it.
`frmGadgetEnterEvent`	Passes the event to the gadget's callback function if the gadget has one. See `FormGadgetHandlerType()`.

Table 11.1 FrmHandleEvent Actions *(continued)*

When FrmHandleEvent receives...	FrmHandleEvent performs these actions...
frmGadgetMiscEvent	Passes the event to the gadget's callback function if the gadget has one. See FormGadgetHandlerType().
frmTitleEnterEvent	Tracks the pen until it is lifted. If it is lifted within the bounds of the form title, adds a frmTitleSelectEvent event to the event queue.
frmTitleSelectEvent	Adds a keyDownEvent with the vchrMenu character to the event queue.
frmUpdateEvent	Calls FrmDrawForm() to redraw the form.
keyDownEvent	Passes the event to the handler for the object that has the focus. If no object has the focus, the event is ignored.
lstEnterEvent	Passes the event and a pointer to the object the event occurred in to LstHandleEvent(). The object pointer is obtained from the event data.
menuCmdBarOpenEvent	Checks if a field object or a table object has the focus and passes the event to the appropriate handler (FldHandleEvent or TblHandleEvent), broadcasts the notification sysNotifyMenuCmdBarOpenEvent, and then displays the command toolbar.
menuEvent	Checks if the menu command is one of the system edit menu commands. The system provides a standard edit menu that contains the commands Undo, Cut, Copy, Paste, Select All, and Keyboard. FrmHandleEvent responds to these commands.

Table 11.1 FrmHandleEvent Actions *(continued)*

When FrmHandleEvent receives...	FrmHandleEvent performs these actions...
penDownEvent; pen position in the bounds of the form object	Checks the list of objects contained by the form to determine if the pen is within the bounds of one. If it is, the appropriate handler is called to handle the event, for example, if the pen is in a control, `CtlHandleEvent` is called. If the pen isn't within the bounds of an object, the event is ignored by the form. If the pen is within the bounds of the help icon, it is tracked until it is lifted, and if it's still within the help icon bounds, the help dialog is displayed.
popSelectEvent	Sets the label of the popup trigger to the current selection of the popup list.
sclEnterEvent or sclRepeatEvent	Passes the event and a pointer to the object the event occurred in to `SclHandleEvent()`.
tblEnterEvent	Passes the event and a pointer to the object the event occurred in to `TblHandleEvent`. The object pointer is obtained from the event data.

Compatibility `FrmHandleEvent` only handles `frmTitleSelectEvent`, `menuCmdBarOpenEvent`, `frmGadgetEnterEvent`, and `frmGadgetMiscEvent` if 3.5 New Feature Set is present. If 5.0 New Feature Set is present, this function should be considered "System Use Only"; applications should do what they can to avoid using it.

See Also `FrmDispatchEvent()`

FrmHelp Function

Purpose Display the specified help message until the user taps the Done button in the help dialog.

Declared In `Form.h`

Prototype

```
void FrmHelp (UInt16 helpMsgId)
```

Parameters

→ *helpMsgId*
Resource ID of help message string.

Returns Returns nothing.

Comments The help message is displayed in a modal dialog that supports scrolling the text if necessary.

FrmHideObject Function

Purpose Erase the specified object and set its attribute data (`usable` bit) so that it does not redraw or respond to the pen.

Declared In `Form.h`

Prototype

```
void FrmHideObject (FormType *formP,
   UInt16 objIndex)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *objIndex*
Index of an object in the form. You can obtain this by using `FrmGetObjectIndex()`.

Returns Returns nothing.

Compatibility Prior to OS version 3.2, this function did not set the `usable` bit of the object attribute data to `false`. On an OS version prior to 3.2 you can work around this bug by directly setting this bit to `false` yourself.

On versions of Palm OS prior to 3.5 this function doesn't affect lists or tables. On Palm OS 3.5 it operates correctly on lists but doesn't have any effect on tables. On Palm OS 4.0 it operates correctly on both lists and tables.

If 3.5 New Feature Set is present and the object is an extended gadget, this function calls the gadget's callback with `formGadgetEraseCmd`. See `FormGadgetHandlerType()`.

See Also `FrmShowObject()`

FrmInitForm Function

Purpose Load and initialize a form resource.

Declared In `Form.h`

Prototype

```
FormType *FrmInitForm (UInt16 rscID)
```

Parameters

→ *rscID*
Resource ID of the form.

Returns Returns a pointer to the form data structure.

When using debug ROMs, `FrmInitForm` displays an error message if the form has already been initialized.

Comments This function does not affect the display (use `FrmDrawForm()` to draw the form) nor make the form active (use `FrmSetActiveForm()` to make it active).

For each initialized form, you must call `FrmDeleteForm` to release the form memory when you are done with the form. Alternatively, you can free the active form by calling `FrmReturnToForm`.

See Also `FrmDoDialog()`, `FrmDeleteForm()`, `FrmReturnToForm()`

FrmNavDrawFocusRing Function

Purpose Tells the system to draw a focus ring for an object.

NOTE: This function is implemented differently on different Palm OS handhelds; developers should use `FrmGlueNavDrawFocusRing()` instead.

Declared In `Form.h`

Prototype

```
Err FrmNavDrawFocusRing (FormType *formP,
    UInt16 objectID, Int16 extraInfo,
    RectangleType *boundsInsideRingP,
    FrmNavFocusRingStyleEnum ringStyle,
    Boolean forceRestore)
```

Parameters

→ *formP*
Pointer to the form containing the object whose ring is to be drawn.

→ *objectID*
ID of object getting the focus ring.

→ *extraInfo*
Any information that you want associated with the focus ring. For example, the system uses this field to store the temporarily-selected item of a focused, embedded list. Pass `frmNavFocusRingNoExtraInfo` if you have no information to associate with the ring.

→ *boundsInsideRingP*
Pointer to a `RectangleType` structure defining the bounds around which the ring should be drawn.

→ *ringStyle*
One of the `FrmNavFocusRingStyleEnum` values that represents the desired style for the ring. Pass `frmNavFocusRingStyleObjectTypeDefault` if you want the default ring style for the object's type.

→ *forceRestore*
Specifies whether, if the ring is already on the object, what is behind the ring should be restored prior to redrawing the ring.

Returns `errNone` if the focus ring was drawn successfully, or an error code such as one of the following otherwise:

`uilibErrInvalidParam`
formP or *boundsInsideRingP* is `NULL`.

`uilibErrInvalidFocusObject`
objectID specifies a control that is not marked as usable.

`uilibErrFormDoesNotHaveFocus`
The specified form doesn't currently have the focus.

`uilibErrObjectFocusModeOff`
The specified form is not in object focus mode.

Compatibility Implemented only if One-Handed Navigation Feature Set is present. Developers should use the Palm OS Glue Library equivalent of this function—`FrmGlueNavDrawFocusRing()`—for maximum compatibility.

See Also `FrmNavGetFocusRingInfo()`, `FrmNavObjectTakeFocus()`, `FrmNavRemoveFocusRing()`

FrmNavGetFocusRingInfo Function

Purpose If a form has a system-drawn focus ring, gets information about the ring.

NOTE: This function is implemented differently on different Palm OS handhelds; developers should use `FrmGlueNavGetFocusRingInfo()` instead.

Declared In `Form.h`

Prototype

```
Err FrmNavGetFocusRingInfo
    (const FormType *formP, UInt16 *objectIDP,
    Int16 *extraInfoP,
    RectangleType *boundsInsideRingP,
    FrmNavFocusRingStyleEnum *ringStyleP)
```

Parameters

→ *formP*
Pointer to the form whose ring information is to be obtained.

← *objectIDP*
Supply a pointer to a `UInt16` that, upon exit will contain the ID of the object with the system-drawn focus ring. Pass `NULL` for this parameter if you don't need this information.

← *extraInfoP*
Supply a pointer to a `UInt16` that, upon exit will contain to any extra information associated with the focus ring; pass `NULL` if you don't need this information. "Extra information" is any information that was associated with the ring when `FrmNavDrawFocusRing()` was called. If there is no extra information associated with the focus ring, `*extraInfoP` is set to `frmNavFocusRingNoExtraInfo`.

← *boundsInsideRingP*
: Supply a pointer to a `RectangleType` structure into which the bounds of the system-drawn focus ring will be written. Pass `NULL` for this parameter if you don't need this information.

← *ringStyleP*
: Supply a pointer to a `FrmNavFocusRingStyleEnum` variable into which the style of the focus ring will be written. Pass `NULL` for this parameter if you don't need this information.

Returns `errNone` if the focus ring was drawn successfully, or an error code such as one of the following otherwise:

`uilibErrFormDoesNotHaveFocus`
: The specified form doesn't currently have the focus.

`uilibErrObjectFocusModeOff`
: The specified form is not in object focus mode.

`uilibErrObjectNotFound`
: There is no object with a system-drawn focus ring.

Comments For embedded lists in interaction mode, the system stores the temporarily selected item in the extra info field.

Compatibility Implemented only if One-Handed Navigation Feature Set is present. Developers should use the Palm OS Glue Library equivalent of this function—`FrmGlueNavGetFocusRingInfo()`—for maximum compatibility.

See Also `FrmNavDrawFocusRing()`, `FrmNavRemoveFocusRing()`

FrmNavObjectTakeFocus Function

Purpose Sends a `frmObjectFocusTakeEvent` event for an object.

NOTE: This function is implemented differently on different Palm OS handhelds; developers should use `FrmGlueNavObjectTakeFocus()` instead.

Declared In `Form.h`

Prototype

```
void FrmNavObjectTakeFocus
    (const FormType *formP, UInt16 objID)
```

Parameters

→ *formP*
Pointer to the form that contains the object.

→ *objID*
ID of the object.

Returns Nothing.

Compatibility Implemented only if One-Handed Navigation Feature Set is present. Developers should use the Palm OS Glue Library equivalent of this function—`FrmGlueNavObjectTakeFocus()`—for maximum compatibility.

FrmNavRemoveFocusRing Function

Purpose Removes the system-drawn ring from around the currently-focused object.

NOTE: This function is implemented differently on different Palm OS handhelds; developers should use `FrmGlueNavRemoveFocusRing()` instead.

Declared In `Form.h`

Prototype

```
Err FrmNavRemoveFocusRing (FormType *formP)
```

Parameters

→ *formP*
Pointer to the form containing the object with the focus ring.

Returns `errNone` if the focus ring was drawn successfully, or an error code such as one of the following otherwise:

`uilibErrInvalidParam`
formP is `NULL`.

`uilibErrFormDoesNotHaveFocus`
The specified form doesn't currently have the focus.

`uilibErrObjectFocusModeOff`
The specified form is not in object focus mode.

Compatibility Implemented only if One-Handed Navigation Feature Set is present. Developers should use the Palm OS Glue Library equivalent of this function—`FrmGlueNavRemoveFocusRing()`—for maximum compatibility.

See Also `FrmNavDrawFocusRing()`, `FrmNavGetFocusRingInfo()`

FrmNewBitmap Function

Purpose Create a new form bitmap dynamically.

Declared In `Form.h`

Prototype

```
FormBitmapType *FrmNewBitmap (FormType **formPP,
    UInt16 ID, UInt16 rscID, Coord x, Coord y)
```

Parameters

↔ *formPP*
: Pointer to a pointer to the form in which the new bitmap is installed. This value is not a handle; that is, the old `formPP` value is not necessarily valid after this function returns because the form may be moved in memory. In subsequent calls, always use the new `formPP` value returned by this function.

→ *ID*
: Symbolic ID of the bitmap, specified by the developer. By convention, this ID should match the resource ID (not mandatory).

→ *rscID*
: Numeric value identifying the resource that provides the bitmap. This value must be unique within the application scope.

→ *x*
: Horizontal coordinate of the upper-left corner of the bitmap's boundaries, relative to the window in which it appears.

→ *y*
: Vertical coordinate of the upper-left corner of the bitmap's boundaries, relative to the window in which it appears.

Returns Returns a pointer to the new bitmap, or 0 if the call did not succeed. The most common cause of failure is lack of memory.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FrmRemoveObject()`

FrmNewForm Function

Purpose Create a new form object dynamically.

Declared In `Form.h`

Prototype

```
FormType *FrmNewForm (UInt16 formID,
   const Char *titleStrP, Coord x, Coord y,
   Coord width, Coord height, Boolean modal,
   UInt16 defaultButton, UInt16 helpRscID,
   UInt16 menuRscID)
```

Parameters

→ *formID*
Symbolic ID of the form, specified by the developer. By convention, this ID should match the resource ID (not mandatory).

→ *titleStrP*
Pointer to a string that is the title of the form.

→ *x*
Horizontal coordinate of the upper-left corner of the form's boundaries, relative to the window in which it appears.

→ *y*
Vertical coordinate of the upper-left corner of the form's boundaries, relative to the window in which it appears.

→ *width*
Width of the form, expressed in pixels. Valid values are 1 - 160.

→ *height*
Height of the form, expressed in pixels.Valid values are 1 - 160.

→ *modal*
`true` specifies that the form ignores pen events outside its boundaries.

→ *defaultButton*
Symbolic ID of the button that provides the form's default action, specified by the developer.

→ *helpRscID*
Symbolic ID of the resource that provides the form's online help, specified by the developer. Only modal dialogs can have help resources.

→ *menuRscID*
Symbolic ID of the resource that provides the form's menus, specified by the developer.

Returns Returns a pointer to the new form object, or 0 if the call did not succeed. The most common cause of failure is lack of memory.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FrmValidatePtr()`, `WinValidateHandle()`, `FrmRemoveObject()`

FrmNewGadget Function

Purpose Create a new gadget dynamically and install it in the specified form.

Declared In `Form.h`

Prototype

```
FormGadgetType *FrmNewGadget (FormType **formPP,
   UInt16 id, Coord x, Coord y, Coord width,
   Coord height)
```

Parameters

↔ *formPP*
Pointer to a pointer to the form in which the new gadget is installed. This value is not a handle; that is, the old `formPP` value is not necessarily valid after this function returns because the form may be moved in memory. In subsequent calls, always use the new `formPP` value returned by this function.

→ *id*
Symbolic ID of the gadget, specified by the developer. By convention, this ID should match the resource ID (not mandatory).

→ *x*
Horizontal coordinate of the upper-left corner of the gadget's boundaries, relative to the window in which it appears.

→ *y*
Vertical coordinate of the upper-left corner of the gadget's boundaries, relative to the window in which it appears.

→ *width*
Width of the gadget, expressed in pixels. Valid values are 1 - 160.

→ *height*
Height of the gadget, expressed in pixels.Valid values are 1 - 160.

Returns Returns a pointer to the new gadget object or 0 if the call did not succeed. The most common cause of failure is lack of memory.

Comments A gadget is a custom user interface object. For more information, see *Palm OS User Interface Guidelines*.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FrmRemoveObject()`

FrmNewGsi Function

Purpose Create a new shift indicator dynamically and install it in the specified form.

Declared In `Form.h`

Prototype

```
FrmGraffitiStateType *FrmNewGsi
    (FormType **formPP, Coord x, Coord y)
```

Parameters

↔ *formPP*
Pointer to a pointer to the form in which the new shift indicator is installed. This value is not a handle; that is, the old `formPP` value is not necessarily valid after this function returns because the form may be moved in memory. In subsequent calls, always use the new `formPP` value returned by this function.

→ *x*
Horizontal coordinate of the upper-left corner of the shift indicator's boundaries, relative to the window in which it appears.

→ *y*
Vertical coordinate of the upper-left corner of the shift indicator's boundaries, relative to the window in which it appears.

Returns Returns a pointer to the new gadget object or 0 if the call did not succeed. The most common cause of failure is lack of memory.

Comments In normal operation, the shift indicator is drawn in the lower-right portion of the screen when the user enters the shift keystroke. You use this function if the shift indicator needs to be drawn in a nonstandard location. For example, the form manager uses it to draw the shift indicator in a custom alert dialog that contains a text field (`FrmCustomResponseAlert()`).

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `FrmRemoveObject()`

FrmNewLabel Function

Purpose Create a new label object dynamically and install it in the specified form.

Declared In `Form.h`

Prototype

```
FormLabelType *FrmNewLabel (FormType **formPP,
    UInt16 ID, const Char *textP, Coord x,
    Coord y,
    FontID font)
```

Parameters

↔ *formPP*
: Pointer to a pointer to the form in which the new label is installed. This value is not a handle; that is, the old `formPP` value is not necessarily valid after this function returns because the form may be moved in memory. In subsequent calls, always use the new `formPP` value returned by this function.

→ *ID*
: Symbolic ID of the label, specified by the developer. By convention, this ID should match the resource ID (not mandatory).

→ *textP*
: Pointer to a string that provides the label text. This string is copied into the label structure.

→ *x*
: Horizontal coordinate of the upper-left corner of the label's boundaries, relative to the window in which it appears.

→ *y*
Vertical coordinate of the upper-left corner of the label's boundaries, relative to the window in which it appears.

→ *font*
Font with which to draw the label text.

Returns Returns a pointer to the new label object or 0 if the call did not succeed. The most common cause of failure is lack of memory.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `CtlValidatePointer()`, `FrmRemoveObject()`

FrmPointInTitle Function

Purpose Check if a coordinate is within the bounds of the form's title.

Declared In `Form.h`

Prototype

```
Boolean FrmPointInTitle (const FormType *formP,
   Coord x, Coord y)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *x, y*
Window-relative x and y coordinates.

Returns Returns `true` if the specified coordinate is in the form's title.

Compatibility Implemented only if 2.0 New Feature Set is present.

FrmPopupForm Function

Purpose Queues a `frmLoadEvent` and a `frmOpenEvent` for the specified form.

Declared In `Form.h`

Prototype

```
void FrmPopupForm (UInt16 formId)
```

Parameters

→ *formID*
Resource ID of form to open.

Returns Returns nothing.

Comments This routine differs from `FrmGotoForm()` in that the current form is not closed. You can call `FrmReturnToForm()` to close a form opened by `FrmPopupForm`.

FrmRemoveObject Function

Purpose Remove the specified object from the specified form.

Declared In `Form.h`

Prototype

```
Err FrmRemoveObject (FormType **formPP,
    UInt16 objIndex)
```

Parameters

↔ *formPP*
: Pointer to a pointer to the form from which this function removes an object. This value is not a handle; that is, the old `formPP` value is not necessarily valid after this function returns. In subsequent calls, always use the new `formPP` value returned by this function.

→ *objIndex*
: The object to remove, specified as an index into the list of objects installed in the form. You can use the `FrmGetObjectIndex()` function to discover this value.

Returns Returns 0 if no error.

Comments You can use this function to remove any form object (a bitmap, control, list, and so on) and free the memory allocated to it within the form data structure. The data structures for most form objects are embedded within the form data structure memory chunk. This function frees that memory and moves the other objects, if necessary, to close up the memory "hole" and decrease the size of the form chunk.

Note that this function does not free memory outside the form data structure that may be allocated to an object, such as the memory allocated to the string in an editable field object.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FrmNewBitmap()`, `FrmNewForm()`, `FrmNewGadget()`, `FrmNewLabel()`, `CtlNewControl()`, `FldNewField()`, `LstNewList()`

FrmRestoreActiveState Macro

Purpose Restores the active window and form state.

Declared In `Form.h`

Prototype

```
#define FrmRestoreActiveState (stateP)
```

Parameters → *stateP*

A pointer to the `FormActiveStateType` structure that you passed to `FrmSaveActiveState` when you saved the state.

Returns Returns zero on success.

Comments Use this function to restore the state of displayed forms to the state that existed before you dynamically showed a new modal form. You must have previously invoked `FrmSaveActiveState()` to save the state.

Compatibility Implemented only if 3.0 New Feature Set is present.

FrmReturnToForm Function

Purpose Erase and delete the currently active form and make the specified form the active form.

Declared In `Form.h`

Prototype

```
void FrmReturnToForm (UInt16 formId)
```

Parameters → *formID*

Resource ID of the form to return to.

Returns Returns nothing.

Comments It is assumed that the form being returned to is already loaded into memory and initialized. Passing a form ID of 0 returns to the first form in the window list, which is the last form to be loaded.

`FrmReturnToForm` does not generate a `frmCloseEvent` when called from a modal form's event handler. It assumes that you have already handled cleaning up your form's variables since you are explicitly calling `FrmReturnToForm`.

See Also `FrmGotoForm()`, `FrmPopupForm()`

FrmSaveActiveState Macro

Purpose Saves the active window and form state.

Declared In `Form.h`

Prototype

```
#define FrmSaveActiveState (stateP)
```

Parameters ↔ *stateP*

A pointer to a `FormActiveStateType` structure that is used to save the state. Pass the same pointer to `FrmRestoreActiveState()` to restore the state. Treat the structure like a black box; that is, don't attempt to read it or write to it.

Returns Returns zero on success.

Comments Use this function to save the state of displayed forms before dynamically showing a new modal form. Invoke `FrmRestoreActiveState()` to restore the state after you remove the modal form.

Compatibility Implemented only if 3.0 New Feature Set is present.

FrmSaveAllForms Function

Purpose Send a `frmSaveEvent` to all open forms.

Declared In `Form.h`

Prototype

```
void FrmSaveAllForms (void)
```

Parameters None.

Returns Returns nothing.

See Also `FrmCloseAllForms()`

FrmSetActiveForm Function

Purpose Set the active form. All input (key and pen) is directed to the active form and all drawing occurs there.

Declared In `Form.h`

Prototype

```
void FrmSetActiveForm (FormType *formP)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

Returns Returns nothing.

Comments A `penDownEvent` outside the form but within the display area is ignored.

Compatibility In Palm OS releases earlier than 3.5, this function generated a `winEnterEvent` for the new form immediately following the `winExitEvent` for the old form. Starting in Palm OS 3.5, `FrmSetActiveForm` does not generate the `winEnterEvent`. The `winEnterEvent` does not occur until the newly active form is drawn.

See Also `FrmGetActiveForm()`

FrmSetCategoryLabel Function

Purpose Set the category label displayed on the title line of a form. If the form's `visible` attribute is set, redraw the label.

Declared In `Form.h`

Prototype

```
void FrmSetCategoryLabel (const FormType *formP,
    UInt16 objIndex, Char *newLabel)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *objIndex*
Index of an object in the form. You can obtain this by using `FrmGetObjectIndex()`.

→ *newLabel*
Pointer to the name of the new category.

Returns Returns nothing.

Comments The pointer to the new label (`newLabel`) is saved in the object.

FrmSetControlGroupSelection Function

Purpose Set the selected control in a group of controls.

Declared In `Form.h`

Prototype

```
void FrmSetControlGroupSelection
    (const FormType *formP, UInt8 groupNum,
    UInt16 controlID)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *groupNum*
Control group number.

→ *controlID*
ID of control to set.

Returns Returns nothing.

Comments This function unsets all the other controls in the group. The display is updated.

NOTE: `FrmGetControlGroupSelection` returns the selection in a control group as an object index, *not* as an object ID, which `FrmSetControlGroupSelection` uses to set the selection.

See Also `FrmGetControlGroupSelection()`

FrmSetControlValue Function

Purpose Sets the current value of a control. If the control is visible, it's redrawn.

Declared In `Form.h`

Prototype

```
void FrmSetControlValue (const FormType *formP,
    UInt16 objIndex, Int16 newValue)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *objIndex*
Index of the control in the form. You can obtain this by using `FrmGetObjectIndex()`.

→ *newValue*
New value to set for the control. For sliders, specify a value between the slider's minimum and maximum. For graphical controls, push buttons, or check boxes, specify 0 for off, nonzero for on.

Returns Returns nothing.

Comments This function works only with graphical controls, sliders, push buttons, and check boxes. If you set the value of any other type of control, the behavior is undefined.

See Also `FrmGetControlValue()`

FrmSetDIAPolicyAttr Function

Purpose Sets a form's dynamic input area policy.

Declared In `Form.h`

Prototype

```
Err FrmSetDIAPolicyAttr (FormPtr formP,
   UInt16 diaPolicy)
```

Parameters

→ *formP*
A pointer to a `FormType` structure.

→ *diaPolicy*
One of the constants listed in "Form Dynamic Input Area Policies" on page 287.

Returns Returns `errNone` if no error or `pinErrInvalidParam` if the `diaPolicy` parameter is out of range.

Comments Applications call this function in response to the `frmLoadEvent` or `frmOpenEvent`, to set the policy that the form should use for opening and closing the dynamic input area. Note that if an application does not call this function, the default is `frmDIAPolicyStayOpen`. This allows legacy application to always be sized appropriately because the input area is always open, with the trigger disabled, while the legacy application is running.

Compatibility Implemented only if the Pen Input Manager Feature Set is present.

See Also `PINSetInputAreaState`, `FrmGetDIAPolicyAttr`

FrmSetEventHandler Function

Purpose Registers the event handler callback routine for the specified form.

Declared In `Form.h`

Prototype

```
void FrmSetEventHandler (FormType *formP,
   FormEventHandlerType *handler)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *handler*
Address of the form event handler function, `FormEventHandlerType()`.

Returns Returns nothing.

Comments `FrmDispatchEvent()` calls this handler whenever it receives an event for a specific form.

`FrmSetEventHandler` must be called right after a form resource is loaded. The callback routine it registers is the mechanism for dispatching events to an application. The tutorial explains how to use callback routines.

FrmSetFocus Function

Purpose Set the focus of a form to the specified object.

Declared In `Form.h`

Prototype

```
void FrmSetFocus (FormType *formP,
   UInt16 fieldIndex)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *fieldIndex*
Index of the object to get the focus in the form. You can obtain this by using `FrmGetObjectIndex()`. You can pass the constant `noFocus` so that no object has the focus.

Returns Returns nothing.

Comments You can set the focus to a field or table object. If the focus is set to a field object, this function turns on the insertion point in the field by calling `FldGrabFocus()` internally.

See Also `FrmGetFocus()`

FrmSetGadgetData Function

Purpose Store a data value in the `data` field of the gadget object.

Declared In `Form.h`

Prototype

```
void FrmSetGadgetData (FormType *formP,
    UInt16 objIndex, const void *data)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *objIndex*
Index of an object in the form. You can obtain this by using `FrmGetObjectIndex()`.

→ *data*
Application-defined value. This value is stored into the `data` field of the gadget data structure (`FormGadgetType`).

Returns Returns nothing.

Comments Gadget objects provide a way for an application to attach custom gadgetry to a form. Typically, the `data` field of a gadget object contains a pointer to the custom object's data structure.

See Also `FrmGetGadgetData()`, `FrmSetGadgetHandler()`

FrmSetGadgetHandler Function

Purpose Registers the gadget event handler callback routine for the specified gadget on the specified form.

Declared In `Form.h`

Prototype

```
void FrmSetGadgetHandler (FormType *formP,
   UInt16 objIndex, FormGadgetHandlerType *attrP)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *objIndex*
Index of a gadget object in the form. You can obtain this by using `FrmGetObjectIndex()`.

→ *attrP*
Address of the callback function. See `FormGadgetHandlerType()`.

Returns Returns nothing.

Comments This function sets the application-defined function that controls the specified gadget's behavior. This function is called when the gadget needs to be drawn, erased, deleted, or needs to handle an event.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `FrmGetGadgetData()`, `FrmSetGadgetData()`

FrmSetMenu Function

Purpose Change a form's menu bar and make the new menu active.

Declared In `Form.h`

Prototype

```
void FrmSetMenu (FormType *formP,
   UInt16 menuRscID)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *menuRscID*
Resource ID of the menu.

Returns Returns nothing.

Compatibility Implemented only if 2.0 New Feature Set is present.

FrmSetNavEntry Function

Purpose Alters the navigation information for a single object in the form.

Declared In `Form.h`

Prototype

```
Err FrmSetNavEntry (FormType *formP,
   UInt16 targetObjectID, UInt16 afterObjectID,
   UInt16 aboveObjectID, UInt16 belowObjectID,
   FrmNavObjectFlagsType objectFlags)
```

Parameters

→ *formP*
Pointer to the form whose navigation order is being changed.

→ *targetObjectID*
The ID of the object which is being edited, added, or removed.

→ *afterObjectID*
The ID of the object which will be to the left of the target object. The object that is currently to the right of the after object will then be to the right of the target object. To place the target object at the beginning of the order, making it the leftmost object in the row, supply an ID of 0. Supply an ID of `frmInvalidObjectId` to have the target object completely removed from the navigation order.

→ *aboveObjectID*
The ID of the object that will be above the target object. Supply an ID of 0 if the target object is in the top row of the form.

→ *belowObjectID*
The ID of the object that will be below the target object. Supply an ID of 0 if the target object is in the bottom row of the form.

→ *objectFlags*
The new object flags for the target object. See `FrmNavObjectFlagsType`.

Returns `errNone` if the operation completed successfully, or an error code otherwise.

Comments

NOTE: This function is not intended to replace the use of navigation resources. A form that needs to have a custom navigation order should *always* provide a navigation resource. This function is mainly for dynamically created forms or forms that have a navigation resource but change the order sometime after form initialization.

Compatibility Implemented only if One-Handed Navigation Feature Set is present.

See Also `FrmGetNavEntry()`

FrmSetNavOrder Function

Purpose Sets the navigation order and/or header information for a form.

Declared In `Form.h`

Prototype

```
Err FrmSetNavOrder (FormType *formP,
    FrmNavHeaderType *navHeaderP,
    FrmNavOrderEntryType *navOrderP)
```

Parameters

→ *formP*
Pointer to the form whose focus order is to be set.

→ *navHeaderP*
Pointer to a `FrmNavHeaderType` structure that contains header information for the form's navigation order. This parameter cannot be `NULL`.

→ *navOrderP*
Pointer to an array of `FrmNavOrderEntryType` structures that specifies the new navigation order for the form. The number of entries in the array must be equal to the header parameter's `numberOfObjects` field. Pass `NULL` if you do not want to change the form's navigation order but only want to change the form's header information.

Returns `errNone` if the operation completed successfully, or an error code otherwise.

Comments This function makes a copy of all information passed-in.

If the object with the current focus is removed from the order, the object will be redrawn in its non-focused state and the form will have no current focus.

NOTE: This function is not intended to replace the use of navigation resources. A form that needs to have a custom navigation order should *always* provide a navigation resource. This function is mainly for dynamically created forms or forms that have a navigation resource but change the order sometime after form initialization.

Compatibility Implemented only if One-Handed Navigation Feature Set is present.

See Also `FrmGetNavOrder()`

FrmSetNavState Function

Purpose Sets the navigation state of a form.

Declared In `Form.h`

Prototype

```
Err FrmSetNavState (FormType *formP,
    FrmNavStateFlagsType stateFlags)
```

Parameters

→ *formP*
Pointer to the form whose navigation state is being set.

→ *stateFlags*
Flags that indicate the new navigation state. This state identifies the form's current focus mode. See "FrmNavStateFlagsType" on page 285 for the flags that make up this parameter.

Returns `errNone` if the operation completed successfully, or an error code otherwise.

Compatibility Implemented only if One-Handed Navigation Feature Set is present.

See Also `FrmGetNavState()`

FrmSetObjectBounds Function

Purpose Set the bounds or position of an object.

Declared In `Form.h`

Prototype

```
void FrmSetObjectBounds (FormType *formP,
    UInt16 objIndex, const RectangleType *bounds)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *objIndex*
Index of an object in the form. You can obtain this by using `FrmGetObjectIndex()`.

→ *bounds*
Window-relative bounds. For the following objects, this sets only the position of the top-left corner: label, bitmap, and shift indicator.

Returns Returns nothing.

Comments Doesn't update the display.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `FrmGetObjectPosition()`, `FrmGetObjectBounds()`, `FrmSetObjectPosition()`

FrmSetObjectPosition Function

Purpose Set the position of an object.

Declared In `Form.h`

Prototype

```
void FrmSetObjectPosition (FormType *formP,
    UInt16 objIndex, Coord x, Coord y)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *objIndex*
Index of an object in the form. You can obtain this by using `FrmGetObjectIndex()`.

→ *x*
Window-relative horizontal coordinate.

→ *y*
Window-relative vertical coordinate.

Returns Returns nothing.

See Also FrmGetObjectPosition(), FrmGetObjectBounds()

FrmSetTitle Function

Purpose Set the title of a form. If the form is visible, draw the new title.

Declared In Form.h

Prototype

```
void FrmSetTitle (FormType *formP,
    Char *newTitle)
```

Parameters

→ *formP*
Pointer to the form object (FormType structure).

→ *newTitle*
Pointer to the new title string.

Returns Returns nothing.

Comments This function draws the title if the form is visible.

This function saves the pointer passed in newTitle; it does *not* make a copy. The value of newTitle must not be a pointer to a stack-based object.

Compatibility Earlier versions of this function redrew the title without first erasing the old one. This problem was corrected in version 3.0 of Palm OS.

See Also FrmGetTitle(), FrmCopyTitle(), FrmCopyLabel()

FrmShowObject Function

Purpose Set a form object as usable. If the form is visible, draw the object.

Declared In Form.h

Prototype

```
void FrmShowObject (FormType *formP,
    UInt16 objIndex)
```

Parameters

→ *formP*
Pointer to the form object (FormType structure).

→ *objIndex*
Index of an object in the form. You can obtain this by using `FrmGetObjectIndex()`.

Returns Returns nothing.

Compatibility On versions of Palm OS prior to 3.5 this function doesn't affect lists or tables. On Palm OS 3.5 it operates correctly on lists but doesn't have any effect on tables. On Palm OS 4.0 it operates correctly on both lists and tables.

If 3.5 New Feature Set is present and the object is an extended gadget, this function calls the gadget's callback with `formGadgetDrawCmd`. See `FormGadgetHandlerType()`.

See Also `FrmHideObject()`

FrmUpdateForm Function

Purpose Send a `frmUpdateEvent` to the specified form.

Declared In `Form.h`

Prototype

```
void FrmUpdateForm (UInt16 formId,
   UInt16 updateCode)
```

Parameters

→ *formId*
Resource ID of form to update.

→ *updateCode*
An application-defined code that can be used to indicate what needs to be updated. Specify the code frmRedrawUpdateCode to indicate that the whole form should be redrawn.

Returns Returns nothing.

Comments If the `frmUpdateEvent` posted by this function is handled by the default form event handler, `FrmHandleEvent()`, the `updateCode` parameter is ignored. `FrmHandleEvent` always redraws the form.

If you handle the `frmUpdateEvent` in a custom event handler, you can use the `updateCode` parameter any way you want. For example, you might use it to indicate that only a certain part of the form needs to be redrawn. If you do handle the `frmUpdateEvent`,

be sure to return `true` from your event handler so that the default form handler does not also redraw the whole form.

If you do handle the `frmUpdateEvent` in a custom event handler, be sure to handle the case where `updateCode` is set to `frmRedrawUpdateCode`, and redraw the whole form. This event (and code) is sent by the system when the whole form needs to be redrawn because the display needs to be refreshed.

FrmUpdateScrollers Function

Purpose Visually update (show or hide) the field scroll arrow buttons.

Declared In `Form.h`

Prototype

```
void FrmUpdateScrollers (FormType *formP,
   UInt16 upIndex, UInt16 downIndex,
   Boolean scrollableUp, Boolean scrollableDown)
```

Parameters

→ *formP*
Pointer to the form object (`FormType` structure).

→ *upIndex*
Index of the up-scroller button. You can obtain this by using `FrmGetObjectIndex()`.

→ *downIndex*
Index of the down-scroller button. You can obtain this by using `FrmGetObjectIndex`.

→ *scrollableUp*
Set to true to make the up scroll arrow active (shown), or `false` to hide it.

→ *scrollableDown*
Set to true to make the down scroll arrow active (shown), or `false` to hide it.

Returns Returns nothing.

Comments This function was originally only supposed to be used with textual buttons, because it uses the Symbol font to update the arrow appearance (either black or gray, up or down). However, it is somewhat tolerant of various button types. On non-textual buttons you can get the enable/disable behavior, which, if you're using a gadget, you can use to change the appearance of your custom

scroller button. If you don't pass this function a textual button, it won't try to update the button's appearance using the button's the "text" member.

FrmValidatePtr Function

Purpose Return `true` if the specified pointer references a valid form.

Declared In `Form.h`

Prototype
```
Boolean FrmValidatePtr (const FormType *formP)
```

Parameters → *formP*
Pointer to be tested.

Returns Returns `true` if the specified pointer is a non-`NULL` pointer to an object having a valid form structure.

Comments This function is intended for debugging purposes only. Do not include it in released code.

To distinguish between a window and a form in released code, instead of using this function, look at the flag `windowFlags.dialog` in the `WindowType` structure. This flag is `true` if the window is a form.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `ECFrmValidatePtr()`

FrmVisible Function

Purpose Return `true` if the form is visible (is drawn).

Declared In `Form.h`

Prototype
```
Boolean FrmVisible (const FormType *formP)
```

Parameters → *formP*
Pointer to the form object (`FormType` structure).

Returns Returns `true` if the form is visible; `false` if it is not visible.

See Also `FrmDrawForm()`, `FrmEraseForm()`

Application-Defined Functions

FormCheckResponseFuncType Function

Purpose Callback function for `FrmCustomResponseAlert()`.

Declared In `Form.h`

Prototype

```
Boolean FormCheckResponseFuncType (Int16 button,
   Char *attempt)
typedef FormCheckResponseFuncType
   *FormCheckResponseFuncPtr
```

Parameters

→ *button*
The ID of the button that the user tapped.

→ *attempt*
The string that the user entered in the alert dialog.

Returns Return `true` if the dialog should be dismissed. Return `false` if the dialog should not be dismissed.

Comments This function is called at these times during the `FrmCustomResponseAlert` routine:

- At the beginning of `FrmCustomResponseAlert`, this function is called with a button ID of `frmResponseCreate`. This constant indicates that the dialog is about to be displayed, and your function should perform any necessary initialization. For example, on a Japanese system, a password dialog might need to disable the Japanese FEP. So it would call `TsmSetFepMode(NULL, tsmFepModeOff)` in this function.
- When the user has tapped a button on the dialog. The function should process the `attempt` string. If the string is valid input, the function should return `true`. If not, it should return `false` to give the user a chance to re-enter the string.
- At the end of `FrmCustomResponseAlert`, this function is called with a button ID of `frmResponseQuit`. This gives the callback a change to perform any cleanup, such as re-enabling the Japanese FEP.

Compatibility Implemented only if 3.5 New Feature Set is present.

FormEventHandlerType Function

Purpose The event handler callback routine for a form.

Declared In `Form.h`

Prototype

```
Boolean FormEventHandlerType (EventType *eventP)
typedef FormEventHandlerType *FormEventHandlerPtr
```

Parameters

→ *eventP*
Pointer to the form event (`FormType` structure).

Returns Must return `true` if this routine handled the event, otherwise `false`.

Comments `FrmDispatchEvent()` calls this handler whenever it receives an event for the form.

This callback routine is the mechanism for dispatching events to particular forms in an application. The callback is registered by the routine `FrmSetEventHandler()`.

FormGadgetHandlerType Function

Purpose The event handler callback for an extended gadget.

Declared In `Form.h`

Prototype

```
Boolean (FormGadgetHandlerType)
    (struct FormGadgetTypeInCallback *gadgetP,
    UInt16 cmd, void *paramP)
```

Parameters

→ *gadgetP*
Pointer to the gadget structure. See `FormGadgetType`.

→ *cmd*
A constant that specifies what action the handler should take. This can be one of the following:

`formGadgetDeleteCmd`
Sent by `FrmDeleteForm()` to indicate that the gadget is being deleted and must clean up any memory it has allocated or perform other cleanup tasks.

`formGadgetDrawCmd`
Sent by `FrmDrawForm()` and `FrmShowObject()` to indicate that the gadget must be drawn or redrawn.

`formGadgetEraseCmd`
Sent by `FrmHideObject()` to indicate that the gadget is going to be erased. `FrmHideObject` clears the `visible` and `usable` flags for you. If you return `false`, it also calls `WinEraseRectangle` to erase the gadget's bounds.

`formGadgetHandleEventCmd`
Sent by `FrmHandleEvent()` to indicate that a gadget event has been received. The `paramP` parameter contains the pointer to the `EventType` structure.

→ *paramP*
`NULL` except if `cmd` is `formGadgetHandleEventCmd`. In that case, this parameter holds the pointer to the `EventType` structure containing the event.

Returns Return `true` if the event was handed successfully; `false` otherwise.

Comments If this function performs any drawing in response to the `formGadgetDrawCmd`, it should set the gadget's `visible` attribute flag. (`gadgetP->attr.visible = true`). This flag indicates that the gadget appears on the screen. If you don't set the `visible` flag, the gadget won't be erased when `FrmHideObject()` is called. (`FrmHideObject` immediately returns if the object's `visible` flag is `false`.)

Note that if the function receives the `formGadgetEraseCmd`, it may simply choose to perform any necessary cleanup and return `false`. If the function returns `false`, `FrmHideObject` erases the gadget's bounding rectangle. If the function returns `true`, it must erase the gadget area itself.

If this function receives a `formGadgetHandleEventCmd`, `paramP` points one of two events: `frmGadgetEnterEvent` or `frmGadgetMiscEvent`. The `frmGadgetEnterEvent` is passed when there is a `penDownEvent` within the gadget's bounds. This function should track the pen and perform any necessary highlighting. The `frmGadgetMiscEvent` is never sent by the

system. Your application may choose to use it if at any point it needs to send data to the extended gadget. In this case, the event has one or both of these fields defined: `selector`, an unsigned integer, and `dataP`, a pointer to data.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `FrmSetGadgetHandler()`

12

Graffiti Shift

This chapter provides reference material for the shift indicator API, declared in the header file `GraffitiShift.h`.

GraffitiShift Functions

GsiEnable Function

Purpose Enable or disable the shift indicator.

Declared In `GraffitiShift.h`

Prototype

```
void GsiEnable (const Boolean enableIt)
```

Parameters

`enableIt`
: `true` to enable, `false` to disable.

Returns Returns nothing.

Comments Enabling the indicator makes it visible, disabling it makes the insertion point invisible.

GsiEnabled Function

Purpose Return `true` if the shift indicator is enabled, or `false` if it's disabled.

Declared In `GraffitiShift.h`

Prototype

```
Boolean GsiEnabled (void)
```

Parameters `None.`

Returns `true` if enabled, `false` if not.

GsiInitialize Function

Purpose Initialize the global variables used to manage the shift indicator.

Declared In `GraffitiShift.h`

Prototype
```
void GsiInitialize (void)
```

Parameters `None.`

Returns Returns nothing.

GsiSetLocation Function

Purpose Set the display-relative position of the shift indicator.

Declared In `GraffitiShift.h`

Prototype
```
void GsiSetLocation (const Int16 x, const Int16 y)
```

Parameters

`x, y`
: Coordinate of left side and top of the indicator.

Returns Returns nothing.

Comments The indicator is not redrawn by this routine.

Do not use this function in application code. It is used internally by the Form Manager. If you need to change the shift indicator's location to support devices with a dynamic input area, see knowledge base article 507 for sample code that shows you how to do so.

GsiSetShiftState Function

Purpose Set the shift indicator.

Declared In `GraffitiShift.h`

Prototype
```
void GsiSetShiftState (const UInt16 lockFlags,
   const UInt16 tempShift)
```

Parameters

`lockFlags`
: `glfCapsLock` or `glfNumLock`.

`tempShift`
: The current temporary shift.

Returns Returns nothing.

Comments This function affects only the state of the UI element, not the underlying Graffiti® or Graffiti 2 engine.

See Also `GrfSetState()`

13

Insertion Point

This chapter provides reference material for the insertion point API, declared in the header file `InsPoint.h`.

For more information on the insertion point, see the section "Insertion Point" in the *Palm OS Programmer's Companion*, vol. I.

Insertion Point Functions

InsPtEnable Function

Purpose Enable or disable the insertion point. When the insertion point is disabled, it's invisible; when it's enabled, it blinks.

Declared In `InsPoint.h`

Prototype

```
void InsPtEnable (Boolean enableIt)
```

Parameters

`enableIt`
: `true` = enable; false = disable

Returns Returns nothing.

Comments This function is called by the Form functions when a text field loses or gains the focus, and by the Windows function when a region of the display is copied (`WinCopyRectangle()`).

Compatibility If Palm OS Cobalt Feature Set is present this function is unimplemented.

See Also InsPtEnabled()

InsPtEnabled Function

Purpose Return `true` if the insertion point is enabled or `false` if the insertion point is disabled.

Declared In `InsPoint.h`

Prototype
```
Boolean InsPtEnabled (void)
```

Parameters `None.`

Returns Returns `true` if the insertion point is enabled (blinking); returns `false` if the insertion point is disabled (invisible).

Compatibility If Palm OS Cobalt Feature Set is present this function is unimplemented.

See Also InsPtEnable()

InsPtGetHeight Function

Purpose Return the height of the insertion point.

Declared In `InsPoint.h`

Prototype
```
Int16 InsPtGetHeight (void)
```

Parameters `None.`

Returns Returns the height of the insertion point, in pixels.

Compatibility If Palm OS Cobalt Feature Set is present this function is unimplemented.

InsPtGetLocation Function

Purpose Return the screen-relative position of the insertion point.

Declared In `InsPoint.h`

Prototype
```
void InsPtGetLocation (Int16 *x, Int16 *y)
```

Parameters `x, y`
Pointer to top-left position of insertion point's x and y coordinate.

Returns Returns nothing. Stores the location in `x` and `y`.

Comments This function is called by the Field functions. An application would not normally call this function.

Compatibility If Palm OS Cobalt Feature Set is present this function is unimplemented.

InsPtSetHeight Function

Purpose Set the height of the insertion point.

Declared In `InsPoint.h`

Prototype

```
void InsPtSetHeight (const Int16 height)
```

Parameters

`height`
Height of the insertion point in pixels.

Returns Returns nothing.

Comments Set the height of the insertion point to match the character height of the font used in the field that the insertion point is in. When the current font is changed, the insertion point height should be set to the line height of the new font.

If the insertion point is visible when its height is changed, it's erased and redrawn with its new height.

Compatibility If Palm OS Cobalt Feature Set is present this function is unimplemented.

See Also InsPtGetHeight()

InsPtSetLocation Function

Purpose Set the screen-relative position of the insertion point.

Declared In `InsPoint.h`

Prototype

```
void InsPtSetLocation (const Int16 x,
    const Int16 y)
```

Parameters

`x, y`
Number of pixels from the left side (top) of the display.

Returns Returns nothing.

Comments The position passed to this function is the location of the top-left corner of the insertion point.

This function should be called only by the Field functions.

Compatibility If Palm OS Cobalt Feature Set is present this function is unimplemented.

See Also `InsPtGetLocation()`

14

Lists

This chapter provides information about list objects by discussing these topics:

- List Data Structures
- List Resources
- List Functions
- Application-Defined Function

The header file `List.h` declares the API that this chapter describes. For more information on lists, see the section "Lists" in the *Palm OS Programmer's Companion*, vol. I.

List Data Structures

ListAttrType Struct

Purpose The `ListAttrType` bit field defines the visible characteristics of the list.

Prototype

```
typedef struct {
  UInt16 usable:1;
  UInt16 enabled:1;
  UInt16 visible:1;
  UInt16 poppedUp:1;
  UInt16 hasScrollBar:1.
  UInt16 search:1;
  UInt16 reserved:2;
} ListAttrType;
```

Fields

`usable`

Set to make the list usable.

If not set, the list is not considered part of the current interface of the application, and does not appear on screen.

`enabled`
: Not used.

`visible`
: Set when the list object is drawn, and cleared when the list object is erased.

  This attribute is set and cleared internally.

`poppedUp`
: Set to indicate that the choices are displayed in a popup window.

  This attribute is set and cleared internally.

`hasScrollBar`
: Set to indicate that the list has a scroll bar.

`search`
: Set to enable incremental search. If incremental search is enabled, when the list is displayed the user can navigate the list by entering up to five characters. The list will scroll to present the first list item that matches the entered characters. This feature only works for popup lists, and only works if the list is sorted and the list items are available to the List Manager (that is, you don't pass `NULL` to `LstSetListChoices()`).

`reserved`
: Reserved for system use.

ListType Struct

Purpose The `ListType` structure is defined below.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `ListType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  UInt16 id;
  RectangleType bounds;
  ListAttrType attr;
  Char **itemsText;
  Int16 numItems;
  Int16 currentItem;
  Int16 topItem;
  FontID font;
  UInt8 reserved;
  WinHandle popupWin;
  ListDrawDataFuncPtr drawItemCallback;
} ListType;
```

Fields

`id`

The ID value, specified by the application developer. This ID value is part of the event data of `lstEnterEvent` and `lstSelectEvent`.

`bounds`

The bounds of the list, relative to the window. For example, to access the bounds of an object in a form whose ID is kObjectID:

```
{
RectangleType rect;
FormPtr formP = FrmGetActiveForm();

FrmGetObjectBounds(formP,
FrmGetObjectIndex(formP, kObjectID),
&rect);
}
```

`attr`

The list attributes. See `ListAttrType`.

`itemsText`

A pointer to an array of pointers to the text of the choices. Access with `LstGetSelectionText()`.

For example, to access the string specified by `itemNum` in the list whose ID is `kChoiceList` use the following:

```
{
Char *string;
Int16 itemNum;
...
string =
LstGetSelectionText(GetObjectPtr(kChoicesLis
t), itemNum);
}
```

where `GetObjectPtr` is the following:

```
static void *GetObjectPtr(UInt16 rsrcID){
FormPtr formP;

formP = FrmGetActiveForm();
return FrmGetObjectPtr(formP,
FrmGetObjectIndex(formP, rsrcID));
}
```

If you use a callback routine to draw the list items, note that the `itemsText` pointer you supply to `LstSetListChoices()` is passed to your callback routine. See the comments under `ListDrawDataFuncType()` for tips on using `itemsText` with a callback routine.

`numItems`

The number of choices in the list. Access with `LstGetNumberOfItems()`.

`currentItem`

The currently-selected list choice (0 = first choice). Access with `LstGetSelection()`.

`topItem`
: The first choice displayed in the list. Set this field with `LstSetTopItem()`.

`font`
: The ID of the font used to draw all list text strings.

`reserved`
: Reserved for future use.

`popupWin`
: The handle of the window created when a list is displayed by calling `LstPopupList()`.

`drawItemCallback`
: Function used to draw an item in the list. If `NULL`, the default drawing routine is used instead. Set this field with `LstSetDrawFunction()`. See Application-Defined Function.

List Resources

The List Resource (tLST), and Popup Trigger Resource (tPUT) are used together to represent an active list. See *Palm OS User Interface Guidelines* for more information.

List Functions

LstDrawList Function

Purpose Sets the visible attribute of the list object to `true`, and draws the list object if it is usable.

Declared In `List.h`

Prototype

```
void LstDrawList (ListType *listP)
```

Parameters

→ *listP*
: Pointer to a list object (ListType).

Returns Returns nothing.

Comments If there are more choices than can be displayed, this function ensures that the current selection is visible. The current selection is

highlighted. Note that this function does not ensure the current selection is visible; if you need to do this, call the `LstMakeItemVisible()` function.

If the list is disabled, it's drawn grayed-out (strongly discouraged). If it's empty, nothing is drawn. If it's not usable, nothing is drawn.

See Also `FrmGetObjectPtr()`, `LstPopupList()`, `LstEraseList()`

LstEraseList Function

Purpose Erase a list object.

Declared In `List.h`

Prototype

```
void LstEraseList (ListType *listP)
```

Parameters → *listP*
Pointer to a list object (ListType).

Returns Returns nothing.

Comments The `visible` attribute is set to `false` by this function.

See Also `FrmGetObjectPtr()`, `LstDrawList()`

LstGetNumberOfItems Function

Purpose Return the number of items in a list.

Declared In `List.h`

Prototype

```
Int16 LstGetNumberOfItems (const ListType *listP)
```

Parameters → *listP*
Pointer to a list object (ListType).

Returns Returns the number of items in a list.

See Also `FrmGetObjectPtr()`, `LstSetListChoices()`

LstGetSelection Function

Purpose Return the currently selected choice in the list.

Declared In `List.h`

Prototype

```
Int16 LstGetSelection (const ListType *listP)
```

Parameters

→ *listP*
Pointer to a list object.

Returns Returns the item number of the current list choice. The list choices are numbered sequentially, starting with 0; Returns `noListSelection` if none of the items are selected.

See Also `FrmGetObjectPtr()`, `LstSetListChoices()`, `LstSetSelection()`, `LstGetSelectionText()`

LstGetSelectionText Function

Purpose Return a pointer to the text of the specified item in the list, or `NULL` if no such item exists.

Declared In `List.h`

Prototype

```
Char *LstGetSelectionText (const ListType *listP,
    Int16 itemNum)
```

Parameters

→ *listP*
Pointer to a list object.

→ *itemNum*
Item to select (0 = first item in list).

Returns Returns a pointer to the text of the current selection, or `NULL` if out of bounds.

Comments This is a pointer within ListType, not a copy. This function is only usable if you supplied an array of strings and a count to `LstSetListChoices()`; if your application uses a callback function that dynamically generates the list text, this function returns `NULL`.

See Also `FrmGetObjectPtr()`

LstGetTopItem Function

Purpose Returns the topmost visible item.

Declared In `List.h`

Prototype
```
Int16 LstGetTopItem (const ListType *listP)
```

Parameters → *listP*
Pointer to a list object.

Returns Returns the item number of the top item visible.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `LstGetSelection()`, `LstSetTopItem()`

LstGetVisibleItems Function

Purpose Return the number of visible items.

Declared In `List.h`

Prototype
```
Int16 LstGetVisibleItems (const ListType *listP)
```

Parameters → *listP*
Pointer to a list object.

Returns The number of items visible.

Compatibility Implemented only if 2.0 New Feature Set is present.

LstHandleEvent Function

Purpose Handle event in the specified list; the list object must have its `usable` and `visible` attribute set to `true`.This routine handles two type of events, `penDownEvent` and `lstEnterEvent`; see Comments.

Declared In `List.h`

Prototype
```
Boolean LstHandleEvent (ListType *listP,
    const EventType *eventP)
```

Parameters → *listP*
Pointer to a list object (ListType).

→ *eventP*
Pointer to an EventType structure.

Returns Return `true` if the event was handled. The following cases will result in a return value of `true`:

- A `penDownEvent` within the bounds of the list
- A `lstEnterEvent` with a list ID value that matches the list ID in the list data structure

Comments When this routine receives a `penDownEvent`, it checks if the pen position is within the bounds of the list object. If it is, this routine tracks the pen until the pen comes up. If the pen comes up within the bounds of the list, a `lstEnterEvent` is added to the event queue, and the routine is exited.

When this routine receives a `lstEnterEvent`, it checks that the list ID in the event record matches the ID of the specified list. If there is a match, this routine creates and displays a popup window containing the list's choices and the routine is exited.

If a `penDownEvent` is received while the list's popup window is displayed and the pen position is outside the bounds of the popup window, the window is dismissed. If the pen position is within the bounds of the window, this routine tracks the pen until it comes up. If the pen comes up outside the list object, a `lstEnterEvent` is added to the event queue.

LstMakeItemVisible Function

Purpose Make an item visible, preferably at the top. If the item is already visible, make no changes.

Declared In `List.h`

Prototype

```
void LstMakeItemVisible (ListType *listP,
    Int16 itemNum)
```

Parameters → *listP*
Pointer to a list object (ListType).

→ *itemNum*
Item to select (0 = first item in list).

Returns Returns nothing.

Comments Does *not* visually update the list. You must call `LstDrawList()` to update it.

See Also `FrmGetObjectPtr()`, `LstSetSelection()`, `LstSetTopItem()`, `LstDrawList()`

LstNewList Function

Purpose Create a new list object dynamically and install it in the specified form. This function can be used to create a new popup trigger and its associated list.

Declared In `List.h`

Prototype

```
Err LstNewList (void **formPP, UInt16 id,
    Coord x,
    Coord y, Coord width, Coord height,
    FontID font,
    Int16 visibleItems, Int16 triggerId)
```

Parameters

↔ *formPP*
Pointer to the pointer to the form in which the new list is installed. This value is not a handle; that is, the old `formPP` value is not necessarily valid after this function returns. In subsequent calls, always use the new `formPP` value returned by this function.

→ *id*
Symbolic ID of the list, specified by the developer. By convention, this ID should match the resource ID (not mandatory).

→ *x*
Horizontal coordinate of the upper-left corner of the list's boundaries, relative to the window in which it appears.

→ *y*
Vertical coordinate of the upper-left corner of the list's boundaries, relative to the window in which it appears.

→ *width*
Width of the list, expressed in pixels. Valid values are 1 – 160.

→ *height*
Height of the list, expressed in pixels.Valid values are 1 – 160.

→ *visibleItems*
Number of list items that can be viewed together.

→ *triggerId*
Symbolic ID of the popup trigger associated with the new list (this ID is specified by the developer). A nonzero value for `triggerID` causes this function to create both the list and its associated popup trigger. If the list isn't a popup list, pass 0 for `triggerId`.

Returns Returns 0 if no error.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `LstDrawList()`, `FrmRemoveObject()`

LstPopupList Function

Purpose Display a modal window that contains the items in the list.

Declared In `List.h`

Prototype
```
Int16 LstPopupList (ListType *listP)
```

Parameters → *listP*
Pointer to a list object.

Returns Returns the list item selected, or -1 if no item was selected.

Comments Saves the previously active window. Creates and deletes the new popup window.

See Also FrmGetObjectPtr()

LstScrollList Function

Purpose Scroll the list up or down a number of times.

Declared In `List.h`

Prototype
```
Boolean LstScrollList (ListType *listP,
   WinDirectionType direction, Int16 itemCount)
```

Parameters → *listP*
Pointer to a list object.

→ *direction*
Direction to scroll.

→ *itemCount*
Items to scroll in direction.

Returns Returns `true` when the list is actually scrolled, `false` otherwise. May return `false` if a scroll past the end of the list is requested.

Compatibility Implemented only if 2.0 New Feature Set is present.

LstSetDrawFunction Function

Purpose Set a callback function to draw each item instead of drawing the item's text string.

Declared In `List.h`

Prototype

```
void LstSetDrawFunction (ListType *listP,
    ListDrawDataFuncPtr func)
```

Parameters

→ *listP*
Pointer to a list object.

→ *func*
Pointer to a function that draws items.

Returns Returns nothing.

Comments This function also adjusts `topItem` to prevent a shrunken list from being scrolled down too far. Use this function for custom draw functionality.

See Also `FrmGetObjectPtr()`, `LstGlueGetDrawFunction()`, `LstSetListChoices()`, `ListDrawDataFuncType()`

LstSetHeight Function

Purpose Set the number of items visible in a list.

Declared In `List.h`

Prototype

```
void LstSetHeight (ListType *listP,
    Int16 visibleItems)
```

Parameters

→ *listP*
Pointer to a list object.

→ *visibleItems*
Number of choices visible at once.

Returns Returns nothing.

Comments This function doesn't redraw the list if it's already visible.

See Also FrmGetObjectPtr()

LstSetListChoices Function

Purpose Set the items of a list to the array of text string pointers passed to this function. This functions erases the old list items.

Declared In `List.h`

Prototype

```
void LstSetListChoices (ListType *listP,
    Char **itemsText, Int16 numItems)
```

Parameters

→ *listP*
Pointer to a list object.

→ *itemsText*
Pointer to an array of text strings. See `SysFormPointerArrayToStrings()` for one way to create this array of strings.

→ *numItems*
Number of choices in the list.

Returns Returns nothing.

Comments You need to call the `LstDrawList` function to update the list if it is displayed when you call this function.

NOTE: This function does not copy the strings in the `itemsText` array, which means that you need to ensure that the array is not moved or deallocated until after you are done with the list.

If you use a callback routine to draw the items in your list, the `itemsText` pointer is simply passed to that callback routine and is not otherwise used by the List Manager code. See the comments

under `ListDrawDataFuncType()` for tips on using the `itemsText` parameter with a callback routine.

See Also `FrmGetObjectPtr()`, `LstGlueGetItemsText()`, `LstSetSelection()`, `LstSetTopItem()`, `LstDrawList()`, `LstSetHeight()`, `LstSetDrawFunction()`

LstSetPosition Function

Purpose Set the position of a list.

Declared In `List.h`

Prototype

```
void LstSetPosition (ListType *listP, Coord x,
    Coord y)
```

Parameters

→ *listP*
Pointer to a list object

→ *x*
Left bound.

→ *y*
Top bound.

Returns Returns nothing.

Comments The list is not redrawn. Don't call this function when the list is visible.

See Also FrmGetObjectPtr()

LstSetSelection Function

Purpose Set the selection for a list.

Declared In `List.h`

Prototype

```
void LstSetSelection (ListType *listP,
    Int16 itemNum)
```

Parameters

→ *listP*
Pointer to a list object.

→ *itemNum*
Item to select (0 = first item in list, `noListSelection` = none).

Returns Returns nothing.

Comments The old selection, if any, is unselected. If the list is visible, the selected item is visually updated. The list is scrolled to the selection, if necessary, as long as the list object is both visible and usable.

See Also `FrmGetObjectPtr()`, `LstSetTopItem()`

LstSetTopItem Function

Purpose Set the item visible. The item cannot become the top item if it's on the last page.

Declared In `List.h`

Prototype

```
void LstSetTopItem (ListType *listP,
    Int16 itemNum)
```

Parameters

→ *listP*
Pointer to a list object.

→ *itemNum*
Item to select (0 = first item in list). This must be a valid item number.

Returns Returns nothing.

Comments Does *not* update the display.

NOTE: The value you specify for itemNum must be in the range `0` to max-number-of-items.

See Also `FrmGetObjectPtr()`, `LstGlueGetTopItem`, `LstSetSelection()`, `LstGetTopItem()`, `LstMakeItemVisible()`, `LstDrawList()`, `LstEraseList()`

Application-Defined Function

If you need to perform special drawing for items in the list, call `LstSetDrawFunction()` to set the list drawing callback function. The `ListDrawDataFuncType()` section documents the prototype for the callback function you provide for drawing list items.

ListDrawDataFuncType Function

Purpose Callback function that you provide for drawing items in a list. This function is called whenever the Palm OS needs to draw an element in the list. Your callback function declaration must match the prototype shown here.

Declared In `List.h`

Prototype

```
void ListDrawDataFuncType(Int16 itemNum,
   RectangleType *bounds, Char **itemsText)
```

Parameters

→ *itemNum*
The number of the item to draw.

→ *bounds*
The bounds of the list, relative to the window.

→ *itemsText*
A pointer to an array of pointers to the text of the list items. This is the pointer that you supplied when calling `LstSetListChoices()`.

Returns Returns nothing.

Comments You can call `LstSetDrawFunction()` to register your callback function for the list, which means that your function will be called to draw the list items, rather than using the built-in draw functionality, which displays each item's text string.

Your callback function is called whenever an item in the list needs to be drawn. When it is called, the value of the `itemNum` parameter specifies which item the function is to draw. The `itemsText` parameter, which is what was supplied to `LstSetListChoices`, doesn't actually need to point to a list of strings: you can pass `NULL`, or you can pass a pointer to anything that will be useful to your drawing function. Note, however, that if you pass anything other than a pointer to a list of strings when you call `LstSetListChoices`, you must ensure that `LstGetSelectionText()` is never called since it assumes that this pointer indicates an array of text items. In particular, if your list is associated with a pop-up trigger you must handle the `popSelectEvent` yourself before `FrmHandleEvent()` gets a chance at it since `FrmHandleEvent` calls `LstGetSelectionText`.

WARNING! If the list is a popup list, the form that owns the list is not active while the list is in a window. This means that you cannot call the `FrmGetActiveForm()` function. Instead, use itemsText pointer to access any information that you need for drawing. If you must access the form, use the `FrmGetFormPtr()` function.

Note that the list object manages which colors are used to draw the items and how to draw selected versus unselected items. In almost all circumstances, your drawing function does not have to be concerned with these details.

However, if you do need to determine if the item is selected, you can rely on the fact that the system has set the pen color to one of two colors prior to calling your draw function:

- If the item is selected, the foreground color is `UIObjectSelectedForeground`.
- If the item is not selected, the foreground color is `UIObjectForeground`.

You can determine the foreground color that is in effect for the list item by calling the `WinSetForeColor()` function, which returns the previous value of the foreground color. Remember to call `WinSetForeColor` again to reset the foreground color to what it was. For example:

```
itemColor = WinSetForeColor(myColor)
WinSetForeColor(itemColor)
selectColor = UiColorGetTableEntryIndex(
                   UIObjectSelectedForeground)
if itemColor == selectColor
...
```

See Also `LstSetDrawFunction()`, `UIColorGetTableEntryIndex()`, `WinSetForeColor()`

15

Menus

This chapter describes the menu API as declared in the header file `Menu.h`. It discusses the following topics:

- Menu Data Structures
- Menu Constants
- Menu Resources
- Menu Functions

For more information on menus, see the section "Menus" on page 113 in the *Palm OS Programmer's Companion*, vol. I.

Menu Data Structures

MenuBarAttrType Struct

Purpose The `MenuBarAttrType` bit field defines some characteristics of the menu bar.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `MenuBarAttrType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  UInt16 visible : 1;
  UInt16 commandPending : 1;
  UInt16 insPtEnabled : 1;
  UInt16 needsRecalc : 1;
} MenuBarAttrType;
```

Your code should treat the `MenuBarAttrType` structure as opaque. Use the functions specified in the descriptions below to retrieve and

set each value. Do not attempt to change structure member values directly.

Fields

`visible`
: If set, the menu bar is drawn and visible on the screen. This attribute is set as part of `MenuDrawMenu()`, which is called when the menu is drawn.

`commandPending`
: If set, a menu command shortcut is in progress. This bit is set during `MenuHandleEvent()` if the menu shortcut keystroke is received. If the next key is received before the timeout value is reached, the key is examined to see if it is a valid menu command.

`insPtEnabled`
: Stores the state of the insertion point at the time the menu was drawn so that it can be restored when the menu is erased.

`needsRecalc`
: If set, recalculate menu dimensions.

Compatibility

The `needsRecalc` constant is present only if 3.5 New Feature Set is present.

MenuCmdBarButtonType Struct

Purpose

The `MenuCmdBarButtonType` structure defines a button to be displayed on the command toolbar. The `buttonsData` field of the `MenuCmdBarType` structure contains an array of structures of this type.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `MenuCmdBarButtonType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  UInt16 bitmapId;
  Char name[menuCmdBarMaxTextLength];
  MenuCmdBarResultType resultType;
  UInt8 reserved;
  UInt32 result;
} MenuCmdBarButtonType;
```

Your code should treat the `MenuCmdBarButtonType` structure as opaque. Do not attempt to change structure member values directly. Instead, use `MenuCmdBarAddButton()` to add a button to the display. For the most part, the parameters to `MenuCmdBarAddButton` are the same as the fields in the `MenuCmdBarButtonType`, so there should be no need to alter these fields directly.

`MenuCmdBarGetButtonData()` can be called to access information about command bar buttons.

Fields

`bitmapId`
: Resource ID of the bitmap to display on the button. This bitmap should be 13 pixels high by 16 pixels wide.

`name`
: Text to display in the status message when the user taps the button.

`resultType`
: Specifies what type of data is contained in the `result` field. See `MenuCmdBarResultType`.

`reserved`
: Reserved for future use.

`result`
: Specifies the data to send when the user clicks the button. The data is interpreted as specified by the `resultType` field. The result can be a shortcut character to enqueue in a `keyDownEvent`, a menu item ID to enqueue in a `menuEvent`, or a notification to be broadcast.

Compatibility

This structure is defined only if 3.5 New Feature Set is present.

MenuCmdBarResultType Enum

Purpose The `MenuCmdBarResultType` enum specifies how the `result` field in the `MenuCmdBarButtonType` structure should be interpreted.

Prototype

```
typedef enum {
  menuCmdBarResultNone,
  menuCmdBarResultChar,
  menuCmdBarResultMenuItem,
  menuCmdBarResultNotify
} MenuCmdBarResultType;
```

Constants

`menuCmdBarResultNone`
: Send nothing.

`menuCmdBarResultChar`
: The result is a character to send in a `keyDownEvent`.

`menuCmdBarResultMenuItem`
: The result is the ID of the menu item to send in a `menuEvent`.

`menuCmdBarResultNotify`
: The result is a notification constant to be broadcast using SysNotifyBroadcastDeferred()

Compatibility This enum is defined only if 3.5 New Feature Set is present.

MenuCmdBarType Struct

Purpose The `MenuCmdBarType` structure defines the command toolbar. This command toolbar is allocated and displayed when the user draws the shortcut stroke in the input area. It is deallocated when `MenuEraseStatus()` is called, which occurs most frequently when the timeout value has elapsed.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `MenuCmdBarType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct MenuCmdBarType {
  WinHandle bitsBehind;
  Int32 timeoutTick;
  Coord top;
  Int16 numButtons;
  Boolean insPtWasEnabled;
  Boolean gsiWasEnabled;
  Boolean feedbackMode;
  MenuCmdBarButtonType *buttonsData;
} MenuCmdBarType;
```

Your code should treat the `MenuCmdBarType` structure as opaque. Do not attempt to change structure member values directly.

Fields

`bitsBehind`
: Handle for the window that contains the region obscured by the command toolbar.

`timeoutTick`
: Timeout value given in system ticks. If the user hasn't specified a command after this many ticks, the command toolbar is erased from the screen and deallocated from memory. This value also specifies how long the status message is displayed after the user successfully enters a command.

`top`
: The top bound of the command toolbar given in display-relative coordinates. The command toolbar is as wide as the screen and displays at the bottom of the screen.

`numButtons`
: Number of buttons displayed on the command toolbar.

`insPtWasEnabled`
: If `true`, the insertion point was enabled before the command toolbar was displayed and should be re-enabled when the command toolbar is erased. If `false`, the insertion point was disabled.

`gsiWasEnabled`
: If `true`, the shift indicator was enabled before the command toolbar was displayed and should be re-enabled when the command toolbar is erased. If `false`, the shift indicator was disabled.

`feedbackMode`
: If `true`, the command toolbar is currently displaying a status message. The status message is displayed to tell the user what command is being performed. If `false`, the command toolbar is awaiting input.

`buttonsData`
: The list of buttons to display on the command toolbar. See `MenuCmdBarButtonType`. Buttons are stored in this list sequentially with the rightmost button at index 0. Access with `MenuCmdBarGetButtonData()`.

Compatibility This structure is defined only if 3.5 New Feature Set is present.

MenuBarPtr Typedef

Purpose The `MenuBarPtr` type defines a pointer to a `MenuBarType`.

Prototype

```
typedef MenuBarType *MenuBarPtr;
```

MenuBarType Struct

Purpose The `MenuBarType` structure defines the menu bar. There is one menu bar per form.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `MenuBarType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  WinHandle barWin;
  WinHandle bitsBehind;
  WinHandle savedActiveWin;
  WinHandle bitsBehindStatus;
  MenuBarAttrType attr;
  Int16 curMenu;
```

```
  Int16 curItem;
  Int32 commandTick;
  Int16 numMenus;
  MenuPullDownPtr menus;
} MenuBarType;
```

Your code should treat the `MenuBarType` structure as opaque. Do not attempt to change structure member values directly.

Fields

`barWin`
: Handle for the window that contains the menu bar.

`bitsBehind`
: Handle for the window that contains the region obscured by the menu bar.

`savedActiveWin`
: Handle where the currently active window is saved so that it can be restored when the menu is erased.

`bitsBehindStatus`
: Handle where the bits behind the status message are saved so that when the message display terminates, the bits can be restored.

  The status message is displayed when the user activates the menu through the use of the command keystroke.

`attr`
: Menu bar attributes. See `MenuBarAttrType`.

`curMenu`
: Menu number for the currently visible menu. Menus are numbered sequentially starting with 0. The value is preserved when the menu bar is dismissed. A value of `noMenuSelection` indicates that there is no current pull-down menu.

`curItem`
: Item number of the currently highlighted menu item. The items in each menu are numbered sequentially, starting with zero.

  A value of `noMenuItemSelection` indicates that there is no current item selected.

`commandTick`
: Tick count at which the status message should be erased.

`numMenus`
: Number of pull-down menus on the menu bar.

`menus`
: Array of `MenuPullDownType` structures.

Compatibility If 3.5 New Feature Set is present, the `bitsBehindStatus` and `commandTick` fields are defined but are not used. Instead, the `bitsBehind` and `timeoutTick` fields in `MenuCmdBarType` define the save-behind window and the timeout value for the command toolbar.

MenuItemType Struct

Purpose The `MenuItemType` structure defines a specific item within a menu. The `items` array in the `MenuPullDownType` structure contains one `MenuItemType` structure for each menu item in the pull-down menu.

If 3.5 New Feature Set is present, you can add a menu item to a pull-down menu programmatically using `MenuAddItem()`.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `MenuItemType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  UInt16 id;
  Char command;
  UInt8 hidden: 1;
  UInt8 reserved: 7;
  Char *itemStr;
} MenuItemType;
```

Fields

`id`
: ID value you specified when you created the menu item. This ID value is included as part of the event data of a `menuEvent`.

`command`
: Shortcut key. If you provide shortcuts, make sure that each shortcut is unique among all commands available at that time.

`hidden`
: If `true`, the menu item is hidden. If `false`, it is displayed. You can set and clear this value using `MenuHideItem()` and `MenuShowItem()`.

`reserved`
: Reserved for future use.

`itemStr`
: Pointer to the text to display for this menu item, including the shortcut key. To include a shortcut key, begin the string with the item's text, then type a tab character, and then the item's shortcut key.

  To create a separator bar, create a one-character string containing the `MenuSeparatorChar` constant.

Compatibility The `hidden` and `reserved` fields are defined only if 3.5 New Feature Set is present.

MenuPullDownPtr Typedef

Purpose The `MenuPullDownPtr` type defines a pointer to a `MenuPullDownType`.

Prototype

```
typedef MenuPullDownType *MenuPullDownPtr;
```

MenuPullDownType Struct

Purpose The `MenuPullDownType` structure defines a specific menu accessed from the menu bar. The menus array in the `MenuBarType` structure contains one `MenuPullDownType` structure for each pull-down menu associated with the menu bar.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `MenuPullDownType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  WinHandle menuWin;
  RectangleType bounds;
  WinHandle bitsBehind;
  RectangleType titleBounds;
  Char *title;
  UInt16 hidden : 1;
  UInt16 numItems : 15;
  MenuItemType *items;
} MenuPullDownType;
```

Fields

`menuWin`
: Handle for the window that contains the menu.

`bounds`
: Position and size, in pixels, of the pull-down menu.

`bitsBehind`
: Handle of a window that contains the region obscured by the menu.

`title`
: The menu title (null-terminated string) displayed in the menu bar.

`titleBounds`
: Position and size, in pixels, of the title in the menu bar.

`hidden`
: If `true`, the menu is hidden; if `false`, it is displayed. This field is not currently used.

`numItems`
: Number of items in the menu. Separators count as items.

`items`
: Array of `MenuItemType` structures.

Compatibility

The `hidden` field is defined only if 3.5 New Feature Set is present.

Menu Constants

Constant	Value	Description
`noMenuSelection`	-1	The `curMenu` field of `MenuBarType` is set to this when there is no currently selected menu.
`noMenuItemSelection`	-1	The `curItem` field of `MenuBarType` is set to this when there is no currently selected menu item.
`separatorItemSelection`	-2	The `curItem` field of `MenuBarType` is set to this when a menu separator item is selected.
`MenuSeparatorChar`	'–'	Special character indicating that the menu item is a bar used to separate groups of related menu items. The first character of the `itemStr` string in `MenuItemType` is set to this.

Menu Resources

The menu bar (MBAR) and pull-down menu (MENU) resources are used jointly to represent a menu object on screen. See *Palm OS User Interface Guidelines* for more information.

Menu Functions

MenuAddItem Function

Purpose Adds an item to the currently active menu.

Declared In `Menu.h`

Prototype

```
Err MenuAddItem (UInt16 positionId, UInt16 id,
    Char cmd, const Char *textP)
```

Parameters

→ *`positionId`*
ID of an existing menu item. The new menu item is added after this menu item.

→ *id*
: ID value to use for the new menu item.

→ *cmd*
: Shortcut key. If you provide shortcuts, make sure that each shortcut is unique among all commands available at that time.

→ *textP*
: Pointer to the text to display for this menu item, including the shortcut key. To include a shortcut key, begin the string with the item's text, then type a tab character, and then the item's shortcut key.

  To create a separator bar, create a one-character string containing the `MenuSeparatorChar` constant.

Returns Returns 0 upon success or one of the following if an error occurs:

`menuErrNoMenu`
: The `textP` parameter is `NULL`.

`menuErrSameId`
: The menu already contains an item with the ID `id`.

`menuErrNotFound`
: The menu doesn't contain an item with the ID `positionId`.

May display a fatal error message if there is no current menu.

Comments This function creates a new `MenuItemType` structure and adds it to the `MenuBarType`'s item list.

You should call this function only in response to a `menuOpenEvent`. This event is generated when the menu is first made active. In general, a form's menu becomes active the first time a `keyDownEvent` with a `vchrMenu` or `vchrCommand` is generated, and it remains active until a new form (including a modal form or alert panel) is displayed or until `FrmSetMenu()` is called to change the form's menu. Palm OS® user interface guidelines discourage adding or hiding menu items at any time other than when the menu is first made active.

Compatibility Implemented only if 3.5 New Feature Set is present.

MenuCmdBarAddButton Function

Purpose Defines a button to be displayed on the command toolbar.

Declared In `Menu.h`

Prototype

```
Err MenuCmdBarAddButton (UInt8 where,
   UInt16 bitmapId,
   MenuCmdBarResultType resultType,
   UInt32 result, Char *nameP)
```

Parameters

→ *where*
Either `menuCmdBarOnLeft` to add the button to the left of the other buttons on the command toolbar, `menuCmdBarOnRight` to add it to the right of the other buttons, or a number indicating the exact position of the button. Button positions are numbered from right to left, and the rightmost position is number 1.

→ *bitmapId*
Resource ID of the bitmap to display on the button. The bitmap's dimensions should be 13 pixels high by 16 pixels wide.

→ *resultType*
The type of data contained in the `result` parameter. See `MenuCmdBarResultType`.

→ *result*
The data to send when the user taps this button. This can be a character, a menu item ID, or a notification constant.

→ *nameP*
Pointer to the text to display in the status message if the user taps the button. If `NULL`, the text is taken from the menu item that matches the ID or shortcut character contained in `result`, if a match is found.

If you supply a text buffer for this parameter, `MenuCmdBarAddButton` makes a copy of the buffer.

Returns Returns 0 upon success, or one of the following error codes:

`menuErrOutOfMemory`
There is not enough memory available to perform the operation.

`menuErrTooManyItems`
: The command toolbar already has the maximum number of buttons allowed (currently 8).

Comments

Call this function in response to a `menuCmdBarOpenEvent` or to the notification `sysNotifyMenuCmdBarOpenEvent`. Both of these signal that the user has entered the command keystroke and the command toolbar is about to open. Your response should be to add buttons to the toolbar and to return `false`, indicating that you have not completely handled the event.

The `sysNotifyMenuCmdBarOpenEvent` notification is intended to be used only by shared libraries, system extensions, and other code resources that do not use an event loop. If you're writing an application, always respond to the event instead of the notification; an application should only add buttons to the toolbar if it is the current application. If you register for the notification, you receive it each time the command toolbar is displayed, whether your application is active or not.

Note that the command toolbar is allocated each time it is opened and is deallocated when it is erased from the screen.

There is a limited amount of space in which to display buttons on the command toolbar. You should limit the number of buttons to four or five. The maximum allowed by the system is eight, but you should leave space for the status message that appears after the user chooses an action. Buttons should be contextual; for example, the field code only displays a paste button if there is text on the clipboard. Bitmaps for the buttons should be 16 X 13 pixels.

If a field has focus when the command toolbar is opened, the field manager adds buttons for cut, copy, paste, and undo. If your application does not want this default behavior, set the `preventFieldButtons` field in the `menuCmdBarOpenEvent` structure to `true`. (Note that there is no way to prevent the field buttons from being drawn from within a notification handler.)

The following bitmaps for command toolbar buttons are defined in `UIResources.h`. The system and the built-in applications use these bitmaps to represent the commands listed in the table. Your application should also use them if it performs the same actions. If you use any of these buttons, add them in the order shown from

right to left. (For example, `BarDeleteBitmap`, if used, should always be the rightmost button.)

Bitmap	Command
`BarDeleteBitmap`	Delete record.
`BarPasteBitmap`	Paste clipboard contents at insertion point.
`BarCopyBitmap`	Copy selection.
`BarCutBitmap`	Cut selection.
`BarUndoBitmap`	Undo previous action.
`BarSecureBitmap`	Show Security dialog.
`BarBeamBitmap`	Beam current record.
`BarInfoBitmap`	Show Info dialog (Launcher).

It is best to add buttons on the left side. If you add buttons to the right, this function moves all existing buttons over one position to the left. You can also specify an exact position for the `where` parameter. The positions are numbered from right to left with the rightmost position being 1. If you specify an exact position, the function leaves space for the other buttons. For example, if you specify position 3 and there are no buttons displayed at positions 1 and 2, there will be blank spots to the right of your button.

The `result` and `resultType` parameters specify what the result should be if the user taps the button. `result` contains the actual data, and `resultType` contains a constant that specifies the type of data in `result`. Typically, the result is to enqueue a `menuEvent`. In this case, `resultType` is `menuCmdBarResultMenuItem` and the `result` is the ID of the menu item that should included in the event.

You may also specify the shortcut character instead of the menu ID; however, doing so is inefficient. When `result` is a shortcut character, the `MenuHandleEvent()` function enqueues a `keyDownEvent` with the character in `result`. During the next cycle of the event loop, `MenuHandleEvent` enqueues a

menuEvent in response to the keyDownEvent. Thus, it is better to have your button enqueue the menuEvent directly.

If you call MenuCmdBarAddButton outside of an application, you might not know of any menu items in the active menu (unless your code has added one using MenuAddItem()). In this case, specify a notification to be broadcast. The notification is broadcast at the top of the next event loop, and it must contain no custom data. (Applications may also use the notification result type.)

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also MenuCmdBarDisplay(), MenuCmdBarGetButtonData()

MenuCmdBarDisplay Function

Purpose Displays the command toolbar.

Declared In Menu.h

Prototype

```
void MenuCmdBarDisplay (void)
```

Parameters None

Returns Returns nothing.

Comments This function displays the command toolbar when the user enters the command keystroke. You normally do not call this function in your own code. The form manager calls it at the end of its handling of menuCmdBarOpenEvent.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also MenuCmdBarAddButton(), MenuCmdBarGetButtonData()

MenuCmdBarGetButtonData Function

Purpose Gets the data for a given command button.

Declared In `Menu.h`

Prototype

```
Boolean MenuCmdBarGetButtonData
    (Int16 buttonIndex, UInt16 *bitmapIdP,
    MenuCmdBarResultType *resultTypeP,
    UInt32 *resultP, Char *nameP)
```

Parameters

→ *buttonIndex*
: Index of the button for which you want to obtain information. Buttons are ordered from right to left, with the rightmost button at index 0.

← *bitmapIdP*
: The resource ID of the bitmap displayed on the button. Pass `NULL` if you don't want to retrieve this value.

← *resultTypeP*
: The type of action this button takes. Pass `NULL` if you don't want to retrieve this value.

← *resultP*
: The result of tapping the button. Pass `NULL` if you don't want to retrieve this information.

← *nameP*
: The text displayed in the status message when this button is tapped. Pass `NULL` if you don't want to retrieve this information. If not `NULL`, `nameP` must point to a string of `menuCmdBarMaxTextLength` size.

Returns Returns `true` if the information was retrieved successfully, `false` if there is no command toolbar or if there is no button at `buttonIndex`.

Comments You can use this function to retrieve information about the buttons that are displayed on the command toolbar. If the command toolbar has not yet been initialized, this function returns `false`.

Note that the command toolbar is allocated when the user enters the command keystroke and deallocated when `MenuEraseStatus()` is called. Thus, the only logical place to call `MenuCmdBarGetButtonData` is in response to a `menuCmdBarOpenEvent` or `sysNotifyMenuCmdBarOpenEvent` notification.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `MenuCmdBarDisplay()`, `MenuCmdBarAddButton()`

MenuDispose Function

Purpose Releases any memory allocated to the menu and the command status and restore any saved bits to the screen.

Declared In `Menu.h`

Prototype

```
void MenuDispose (MenuBarType *menuP)
```

Parameters → *menuP*
Pointer to the menu object to dispose. (See `MenuBarType`.) If `NULL`, this function returns immediately.

Returns Returns nothing.

Comments Most applications do not need to call this function directly. `MenuDispose` is called by the system when the form that contains the menu is no longer the active form, when the form that contains the menu is freed, and when `FrmSetMenu()` is called to change the form's menu bar.

See Also `MenuInit()`, `MenuDrawMenu()`

MenuDrawMenu Function

Purpose Draws the current menu bar and the last pull-down that was visible.

Declared In `Menu.h`

Prototype

```
void MenuDrawMenu (MenuBarType *menuP)
```

Parameters → *menuP*
Pointer to a `MenuBarType`. Must not be `NULL`.

Returns Returns nothing.

Comments Most applications do not need to call this function directly. `MenuHandleEvent()` calls `MenuDrawMenu` when the user taps the Menu silk-screen button (or taps the form's title on Palm OS 3.5 and higher).

The menu bar and the pull-down menu are drawn in front of all the application windows. The state of the insertion point, the bits that are obscured by the menu bar and the pull-down menu, and the currently active window are saved before the menu is drawn. These are all restored when the menu is erased.

A menu keeps track of the last pull-down menu displayed for as long as the menu is active. A menu loses its active status under these conditions:

- When `FrmSetMenu()` is called to change the active menu on the form.
- When a new form, even a modal form or alert panel, becomes active.

Suppose a user selects your application's About item from the Options menu then clicks the OK button to return to the main form. When the About dialog is displayed, it becomes the active form, which causes the main form's menu state to be erased. This menu state is not restored when the main form becomes active again. The next time `MenuDrawMenu` is called (that is, the next time the user taps the Menu silk-screen button), the menu bar is displayed as it was before, and the first pull-down menu listed in the menu bar is displayed instead of the Options pull-down menu.

See Also `MenuInit()`, `MenuDispose()`

MenuEraseStatus Function

Purpose Erases the menu command status.

Declared In `Menu.h`

Prototype

```
void MenuEraseStatus (MenuBarType *menuP)
```

Parameters

→ *menuP*
Pointer to a `MenuBarType`, or `NULL` for the current menu.

Returns Returns nothing.

Comments When the user selects a menu command using the command keystroke, the command toolbar or status message is displayed at the bottom of the screen. `MenuEraseStatus` erases the toolbar or status message.

Under most circumstances, you do not need to call this function explicitly—just let the current menu command status remove itself automatically. Otherwise, you may cause text to be erased before the user has a chance to see it.

You need to call `MenuEraseStatus` explicitly only if the command toolbar covers something that is going to be changed by the menu command the user has selected. For example, if the user selects a command that displays a new form, call `MenuEraseStatus` before executing the command. Also, if the command performs some drawing in the lower portion of the window, call `MenuEraseStatus` before performing the drawing function.

Compatibility

The exact behavior when a menu shortcut character is entered depends on which version of the operating system is running. In versions prior to release 3.5, the system displays the string "Command:" in the lower-left portion of the screen when the user enters the Graffiti® or Graffiti 2 command stroke.

In Palm OS 3.5 and higher, entering the command stroke displays the command toolbar. This toolbar is the entire width of the screen and it displays buttons that the user can tap instead of entering another keystroke. If the user taps a button or enters a character that matches a shortcut character for an item on the active menu, a status message is displayed in the toolbar while the command is executed. Calling `MenuEraseStatus` on Palm OS 3.5 and higher deallocates the command toolbar data structure as well as erasing the command toolbar from the screen.

Because the command toolbar takes up more of the display than the pre-Palm OS 3.5 status message, you may find you need to call `MenuEraseStatus` more frequently in Palm OS 3.5 than in earlier versions.

See Also

`MenuInit()`

MenuGetActiveMenu Function

Purpose Returns a pointer to the currently active menu.

Declared In `Menu.h`

Prototype

```
MenuBarType *MenuGetActiveMenu (void)
```

Parameters `None.`

Returns Returns a pointer to the currently active menu, or `NULL` if there is none.

Comments An active menu is not necessarily visible on the screen. A menu might be active but not visible, for example, if a command shortcut has been entered. In general, a form's menu becomes active the first time a `keyDownEvent` with a `vchrMenu` or `vchrCommand` is generated, and it remains active until a new form (including a modal form or alert panel) is displayed or until `FrmSetMenu()` is called to change the form's menu.

Applications that perform custom drawing to a window often check to see if the menu is visible to ensure that they don't draw on top of the menu. See "Checking Menu Visibility" on page 115 of the *Palm OS Programmer's Companion*, vol. I for instructions on how to verify a menu's visibility.

See Also `MenuHandleEvent()`, `MenuSetActiveMenu()`

MenuHandleEvent Function

Purpose Handles events in the current menu. This routine handles two types of events, `penDownEvent` and `keyDownEvent`.

Declared In `Menu.h`

Prototype

```
Boolean MenuHandleEvent (MenuBarType *menuP,
    EventType *event, UInt16 *error)
```

Parameters

→ *menuP*
Pointer to a MenuBarType data structure.

→ *event*
Pointer to an EventType structure.

→ *error*
Error (or 0 if no error). Currently this function always sets `error` to zero.

Returns Returns `true` if the event is handled; that is, if the event is a `penDownEvent` within the menu bar or the menu, or the event is a `keyDownEvent` that the menu supports. Returns `false` on any other event.

Comments When a `penDownEvent` is received in the menu bar, `MenuHandleEvent` tracks the pen until it comes up. If the pen comes up within the bounds of the menu bar, the selected title is inverted and the appropriate pull-down menu is drawn. Any previous pull-down menu is erased. If the pen comes up outside of the menu bar and pull-down menu, the menu is erased.

When a `penDownEvent` is received in a pull-down menu, `MenuHandleEvent` tracks the pen until it comes up. If the pen comes up within the bounds of the menu, a `menuEvent` containing the resource ID of the selected menu item is added to the event queue. If the pen comes up outside of the bounds of the menu and menu bar, the menu is erased.

If a `keyDownEvent` is received with a `vchrMenu`, the menu is drawn if it is not visible and erased if it is visible.

If a `keyDownEvent` is received with a `vchrCommand`, a command shortcut is in progress. Command shortcuts are handled differently depending on which version of Palm OS is running. On versions earlier than 3.5, the next `keyDownEvent` is checked to see if it is a valid menu shortcut. If so, a `menuEvent` is added to the event queue.

If a `keyDownEvent` is received with a `vchrCommand` on Palm OS 3.5 and higher, `MenuHandleEvent` enqueues a `menuCmdBarOpenEvent` if the command toolbar is not already open. The `menuCmdBarOpenEvent` provides a chance for applications to add their own buttons to the command toolbar. The next event might be either a `keyDownEvent` with a character that completes the shortcut or a `penDownEvent` on one of the buttons on the toolbar. The `keyDownEvent` is processed as with the earlier releases— if it is a valid menu shortcut, a `menuEvent` is added to the event queue. If the next event is a `penDownEvent`, the pen is tracked until it comes up. If the pen comes up within the bounds of

a button, the appropriate action is taken. See the description of `MenuCmdBarAddButton()` for more information.

In Palm OS version 3.5 or higher, if either the `vchrMenu` or the `vchrCommand` event causes a menu to be activated and initialized for the first time, a `menuOpenEvent` containing the reason the menu was initialized (`menuButtonCause` for a `vchrMenu` or `menuCommandCause` for a `vchrCommand`) is added to the event queue, and then the current event is added after it.

MenuHideItem Function

Purpose Makes the specified menu item hidden.

Declared In `Menu.h`

Prototype

```
Boolean MenuHideItem (UInt16 id)
```

Parameters

→ *id*
ID of the menu item to hide.

Returns Returns `true` if the `hidden` attribute of the specified item was successfully enabled, `false` otherwise.

Comments You should call this function only in response to a `menuOpenEvent`. This event is generated when the menu is first made active. In general, a form's menu becomes active the first time a `keyDownEvent` with a `vchrMenu` or `vchrCommand` is generated, and it remains active until a new form (including a modal form or alert panel) is displayed or until `FrmSetMenu()` is called to change the form's menu. Palm OS user interface guidelines discourage adding or hiding menu items at any time other than when the menu is first made active.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `MenuShowItem()`

MenuInit Function

Purpose Loads a menu resource from a resource file.

Declared In `Menu.h`

Prototype

```
MenuBarType *MenuInit (UInt16 resourceId)
```

Parameters → *resourceId*
ID that identifies the menu resource.

Returns Returns the pointer to a memory block allocated to hold the menu resource (a pointer to a `MenuBarType`).

Comments The menu is not usable until `MenuSetActiveMenu()` is called.

Typically, you do not need to call this function directly. A form stores the resource ID of the menu associated with it and initializes that menu as necessary. If you want to change the form's menu, call `FrmSetMenu()`, which handles disposing of the form's current menu, associating the new menu with the form, and initializing when needed.

See Also `MenuSetActiveMenu()`, `MenuDispose()`

MenuSetActiveMenu Function

Purpose Sets the current menu.

Declared In `Menu.h`

Prototype

```
MenuBarType *MenuSetActiveMenu
    (MenuBarType *menuP)
```

Parameters → *menuP*
Pointer to the memory block that contains the new menu, or `NULL` for none.

Returns Returns a pointer to the menu that was active before the new menu was set, or `NULL` if no menu was active.

Comments This function sets the active menu but does not associate it with a form. It's recommended that you call `FrmSetMenu()` instead of `MenuSetActiveMenu`. `FrmSetMenu` sets the active menu, frees the old menu, and associates the newly active menu with the form, which means the menu will be freed when the form is freed.

See Also `MenuGetActiveMenu()`

MenuSetActiveMenuRscID Function

Purpose Sets the current menu by resource ID.

Declared In `Menu.h`

Prototype

```
void MenuSetActiveMenuRscID (UInt16 resourceId)
```

Parameters → *resourceId*
Resource ID of the menu to be made active.

Returns Returns nothing.

Comments This function is similar to `MenuSetActiveMenu()` except that you pass the menu's resource ID instead of a pointer to a menu structure. It is used as an optimization; with `MenuSetActiveMenu`, you must initialize the menu before making it active. Potentially, the application may exit before the menu is needed, making this memory allocation unnecessary. `MenuSetActiveMenuRscID` simply stores the resource ID. The next time a menu is requested, the menu is initialized from this resource.

It's recommended that you call `FrmSetMenu()` instead of calling this function for the reasons given in `MenuSetActiveMenu`.

Compatibility Implemented only if "2.0 New Feature Set" is present.

MenuShowItem Function

Purpose Makes the specified menu item visible.

Declared In `Menu.h`

Prototype

```
Boolean MenuShowItem (UInt16 id)
```

Parameters → *id*
ID of the menu item to display.

Returns Returns `true` if the `hidden` attribute of the specified item was successfully disabled, `false` otherwise.

Comments You should call this function only in response to a `menuOpenEvent`. This event is generated when the menu is first made active. In general, a form's menu becomes active the first time a `keyDownEvent` with a `vchrMenu` or `vchrCommand` is generated, and it remains active until a new form (including a modal form or alert panel) is displayed or until `FrmSetMenu()` is called to change

the form's menu. Palm OS user interface guidelines discourage adding or hiding menu items at any time other than when the menu is first made active.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `MenuHideItem()`

16

Private Records

This chapter describes the private records API as declared in `PrivateRecords.h`. It discusses the following topics:

- Private Record Data Structures
- Private Record Functions

Private Record Data Structures

privateRecordViewEnum Enum

Purpose

The `privateRecordViewEnum` enumerated type provides the available choices for displaying private records.

Prototype

```
typedef enum privateRecordViewEnum {
  showPrivateRecords = 0x00,
  maskPrivateRecords,
  hidePrivateRecords
} privateRecordViewEnum;
```

Constants

`showPrivateRecords`
: Display private records in the user interface.

`maskPrivateRecords`
: Show a shaded rectangle in place of a private record.

`hidePrivateRecords`
: Hide private records and provide no indication in the user interface that they exist.

Private Record Functions

SecSelectViewStatus Function

Purpose Display a form that allows the user to select whether to hide, show, or mask private records.

Declared In `PrivateRecords.h`

Prototype

```
privateRecordViewEnum SecSelectViewStatus (void)
```

Parameters None

Returns Returns a constant that indicates which option the user selected. See `privateRecordViewEnum`.

Comments This function displays a dialog that allows users to change the preference `prefShowPrivateRecords`, which controls how private records are displayed.

When the user taps the OK button in this dialog, `SecVerifyPW()` is called to see if the user changed the preference setting and, if so, to prompt the user to enter the appropriate password.

After calling this function, your code should check the return value or the value of `prefShowPrivateRecords` and mask, display, or hide the private records accordingly. See the description of `TblSetRowMasked()` for a partial example.

Compatibility Implemented only if 3.5 New Feature Set is present.

SecVerifyPW Function

Purpose Display a password dialog, verify the password, and change the private records preference.

Declared In `PrivateRecords.h`

Prototype

```
Boolean SecVerifyPW
    (privateRecordViewEnum newSecLevel)
```

Parameters → *newSecLevel*

The security level (display, hide, or mask) selected on the private records dialog.

Returns Returns `true` if the `prefShowPrivateRecords` preference was successfully changed, `false` if not.

Comments This function checks `newSecLevel` against the current value for the preference. If the two values differ and `newSecLevel` indicates a decrease in security, a dialog is displayed prompting the user to enter a password. (Hidden is considered the most secure, followed by masked. Showing private records is considered the least secure.) If the password is entered successfully, the preference is changed.

This function also displays an alert message if the security level has changed to either hidden or masked.

Compatibility Implemented only if 3.5 New Feature Set is present.

17

Progress Manager

This chapter provides reference material for the Progress Manager.

- Progress Manager Functions
- Application-Defined Functions

The header file `Progress.h` declares the API that this chapter describes. For more information on the progress manager, see the section "Progress Dialogs" in the *Palm OS Programmer's Companion*, vol. I.

Progress Manager Functions

PrgHandleEvent Function

Purpose Handles events related to the active progress dialog.

Declared In `Progress.h`

Prototype

```
Boolean PrgHandleEvent (ProgressPtr prgGP,
   EventType *eventP)
```

Parameters

→ *prgGP*
Pointer to a progress structure created by `PrgStartDialog()`.

→ *eventP*
Pointer to an event. You can pass a `NULL` event to cause this function to immediately call your `PrgCallbackFunc()` function and then update the dialog (for example, after you call `PrgUpdateDialog`).

Returns Returns `true` if the system handled the event. If it returns `false`, you should check if the user canceled the dialog by calling `PrgUserCancel()`.

Comments Use this function instead of `SysHandleEvent()` when you have a progress dialog. `PrgHandleEvent` internally calls `SysHandleEvent` as needed.

Note that the auto power-off feature of the system is automatically disabled when you use this function, unless the dialog is just displaying an error. This function also prevents `appStopEvent` events.

If an update to the dialog is pending (from a call to `PrgUpdateDialog()`, for example) this function handles that event and causes the dialog to be updated. As part of this process, the `textCallback` function you specified in your call to `PrgStartDialog()` is called. Your `textCallback` function should set the `textP` buffer in the `PrgCallbackData` structure with the new message to be displayed in the progress dialog. Optionally, you can also set the `bitmapId` field to include an icon in the update dialog. For more information about the `textCallback` function, see the section "Application-Defined Functions."

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `PrgStartDialog()`, `PrgStopDialog()`, `PrgUpdateDialog()`, `PrgUserCancel()`

PrgStartDialog Function

Purpose Displays a progress dialog that can be updated.

Declared In `Progress.h`

Prototype

```
ProgressPtr PrgStartDialog (const Char *title,
   PrgCallbackFunc textCallback, void *userDataP)
```

Parameters

→ *title*
: Pointer to a title for the progress dialog. Must be a null-terminated string that is no longer than `progressMaxTitle` (20).

→ *textCallback*
: Pointer to a callback function that supplies the text and icons for the current progress state. See `PrgCallbackFunc()`.

→ *userDataP*
A pointer to data that you need to access in the callback function. This address gets passed directly to your function.

Returns A pointer to a progress structure. This pointer must be passed to other progress manager functions and *must* be released by calling `PrgStopDialog()`. `NULL` is returned if the system is unable to allocate the progress structure.

Comments The dialog created by this function can be updated by another process via the `PrgUpdateDialog()` function. The dialog can contain a Cancel or OK button. The initial dialog defaults to stage 0 and calls the `textCallback` function to get the initial text and icon data for the progress dialog.

This function saves the screen bits behind the progress dialog, and these are restored when you call `PrgStopDialog`. Because of this, you should minimize changes to the screen while the progress dialog is displayed, otherwise, the restored bits may not match with what is currently being displayed.

Compatibility This version of the function is available only if 3.2 New Feature Set is present. On earlier systems, `PrgStartDialog` has the prototype shown for `PrgStartDialogV31()`.

See Also `PrgHandleEvent()`, `PrgStopDialog()`, `PrgUpdateDialog()`, `PrgUserCancel()`

PrgStartDialogV31 Function

Purpose Displays a progress dialog that can be updated.

Declared In `Progress.h`

Prototype

```
ProgressPtr PrgStartDialogV31 (const Char *title,
    PrgCallbackFunc textCallback)
```

Parameters

→ *title*
Pointer to a title for the progress dialog. Must be a null-terminated string that is no longer than `progressMaxTitle` (20).

→ *textCallback*
Pointer to a callback function that supplies the text and icons for the current progress state. See `PrgCallbackFunc()`.

Returns A pointer to a progress structure. This pointer must be passed to other progress manager functions and *must* be released by calling `PrgStopDialog()`. `NULL` is returned if the system is unable to allocate the progress structure.

Compatibility This function corresponds to version of `PrgStartDialog()` available on Palm OS® 3.0 and Palm OS 3.1.

See Also `PrgHandleEvent()`, `PrgStopDialog()`, `PrgUpdateDialog()`, `PrgUserCancel()`

PrgStopDialog Function

Purpose Releases memory used by the progress dialog and restores the screen to its initial state.

Declared In `Progress.h`

Prototype

```
void PrgStopDialog (ProgressPtr prgP,
   Boolean force)
```

Parameters

→ *prgP*
Pointer to a progress structure created by `PrgStartDialog()`.

→ *force*
`true` removes the progress dialog immediately, `false` causes the system to wait until the user confirms an error, if one is displayed.

Returns Returns nothing.

Comments If the progress dialog is in a state where it is displaying an error message to the user, this function normally waits for the user to confirm the dialog before it removes the dialog. If you specify `true` for the `force` parameter, this causes the system to remove the dialog immediately.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `PrgHandleEvent()`, `PrgStartDialog()`, `PrgUpdateDialog()`, `PrgUserCancel()`

PrgUpdateDialog Function

Purpose Updates the status of the current progress dialog.

Declared In `Progress.h`

Prototype

```
void PrgUpdateDialog (ProgressPtr prgGP,
   UInt16 err, UInt16 stage,
   const Char *messageP,
   Boolean updateNow)
```

Parameters

→ *prgGP*
Pointer to a progress structure created by `PrgStartDialog()`.

→ *err*
If non-zero, causes the dialog to go into error mode, to display an error message with only an OK button.

→ *stage*
Current stage of progress. The callback function can use this to determine the correct string to display in the updated dialog.

→ *messageP*
Extra text that may be useful in displaying the progress for this stage. Used by the callback function, which can append it to the base message that is based on the stage.

→ *updateNow*
If `true`, the dialog is immediately updated. Otherwise, the dialog is updated on the next call to `PrgHandleEvent()`.

Returns Returns nothing.

Comments For more information about how the parameters are used and the callback function, see the section "Application-Defined Functions."

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also PrgHandleEvent(), PrgStartDialog(), PrgStopDialog(), PrgUserCancel()

PrgUserCancel Macro

Purpose Returns `true` if the user cancelled the process via the progress dialog.

Declared In `Progress.h`

Prototype

```
#define PrgUserCancel (prgP)
```

Parameters → *prgP*
Pointer to a progress structure (`ProgressPtr`) created by `PrgStartDialog()`.

Returns Returns the value of the `cancel` field in the progress structure (as a `UInt16`).

Comments This is a macro you can use to check if the user cancelled the process. If the user did cancel, you can change the progress dialog text to something like "Cancelling," or "Disconnecting," or whatever is appropriate for your application. Then you should cancel the process, end the communication session, or do whatever processing is necessary.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also PrgHandleEvent(), PrgStartDialog(), PrgStopDialog(), PrgUpdateDialog()

Application-Defined Functions

PrgCallbackFunc Function

Purpose Supplies the text and icons for the current progress state.

Declared In `Progress.h`

Prototype

```
Boolean (*PrgCallbackFunc)
    (PrgCallbackDataPtr cbP)
```

Parameters ↔ *cbP*
A pointer to a `PrgCallbackData` structure. See below.

Returns Returns `true` if the progress dialog should be updated using the values you specified in the `PrgCallbackData` structure. If you

specify `false`, the dialog is still updated, but with default status messages. (Returning `false` is not recommended.)

Comments

This is a callback function that you specify when you call `PrgStartDialog()`. The callback function is called by `PrgHandleEvent()` when it needs current progress information to display in the progress dialog.

→ *The system passes this function one parameter, a pointer to a PrgCallbackData structure. Here are the important fields in that data structure (note that indicates you set the field in the textCallback function):*

← *UInt16 stage*
: Current stage (passed from `PrgUpdateDialog`).

↔ *Char *textP*
: Buffer to hold the text to display in the updated dialog. You might want to look up a message in a resource file, based on the value in the `stage` field. Also, you should append the additional text in the `message` field, to form the full string to display. Be sure to include a null terminator at the end of the string you return, and don't exceed the length in `textLen`.

← *UInt16 textLen*
: Maximum length of the text buffer `textP`. Note that this value is set for you by the caller. Be careful not to exceed this length in `textP`.

← *Char *message*
: Additional text to display in the dialog (from the `messageP` parameter to `PrgUpdateDialog`). This should be no longer than `progressMaxMessage` (128).

← *Err error*
: Current error (passed from the `err` parameter to `PrgUpdateDialog`).

→ *UInt16 bitmapId*
: Resource ID of the bitmap to display in the progress dialog, if any.

← *UInt16 canceled:1*
: `true` if user has pressed the cancel button.

← *UInt16 showDetails:1*
: `true` if user pressed the down arrow button on the Palm™ device for more details. (Because this is a non-standard user interface technique, you shouldn't use this feature to display details that users need under normal conditions. It's more for debugging purposes.)

→ *UInt16 textChanged:1*
: If `true`, then update text (defaults to `true`). You can set this to `false` to avoid an update to the text.

← *UInt16 timedOut:1*
: `true` if update caused by a timeout.

↔ *UInt32 timeout*
: Timeout in ticks to force next update. After this number of ticks, an update is automatically triggered (which sets the `timedOut` flag). You can use this feature to do a simple animation effect. Note that you must set the timeout for `EvtGetEvent` to a value that is equal to or less than this value, otherwise you won't get update events as frequently as you expect.

→ *UInt16 delay:1*
: If `true`, delay for one second after updating the dialog. Use this value when you are displaying the final progress message so that users have a chance to see the message before the dialog closes. This field is available only if 3.2 New Feature Set is present.

← *void *userDataP*
: A pointer to any application-defined data that the function needs to access. You specify this pointer as a parameter to `PrgStartDialog()` if the callback function needs to access some application data but does not have access to application globals. This field is available only if 3.2 New Feature Set is present.

In this function, you should set the value of the `textP` buffer to the string you want to display in the progress dialog when it is updated. You can use the value in the `stage` field to look up a message in a string resource. You also might want to append the text in the `message` field to your base string. Typically, the `message` field would contain more dynamic information that depends on a user

selection, such as a phone number, device name, or network identifier, etc.

For example, the `PrgUpdateDialog` function might have been called with a `stage` of 1 and a `messageP` parameter value of a phone number string, "555-1212". Based on the stage, you might find the string "Dialing" in a string resource, and append the phone number, to form the final text "Dialing 555-1212" that you place in the text buffer `textP`.

Keeping the static strings corresponding to various stages in a resource makes it easier to localize your application. More dynamic information can be passed in via the `messageP` parameter to `PrgUpdateDialog`.

NOTE: This function is called only if the parameters passed to `PrgUpdateDialog` have changed from the last time it was called. If `PrgUpdateDialog` is called twice with exactly the same parameters, the `textCallback` function is called only once.

18

Scroll Bars

This chapter provides reference material for the scroll bar API.

- Scroll Bar Data Structures
- Scroll Bar Resources
- Scroll Bar Functions

The header file `ScrollBar.h` declares the API that this chapter describes. For more information on scroll bars, see the section "Scroll Bars" on page 144 in the *Palm OS Programmer's Companion*, vol. I.

Scroll Bar Data Structures

ScrollBarAttrType Struct

Purpose The `ScrollBarAttrType` bit field defines a scroll bar's visible characteristics.

Prototype

```
typedef struct {
  UInt16 usable:1;
  UInt16 visible:1;
  UInt16 hilighted:1;
  UInt16 shown:1;
  UInt16 activeRegion:4;
} ScrollBarAttrType;
```

Fields

`usable`
: If not set, the scroll bar object is not considered part of the current interface of the application, and it doesn't appear on screen.

`visible`
: If set, the scroll bar is allowed to be displayed on the screen. If both `visible` and `shown` are `true`, then the scroll bar is actually displayed on the screen.

`hilighted`
: `true` if either the up arrow or the down arrow is highlighted.

`shown`
: Set if the scroll bar is visible and if `maxValue` > `minValue`. (See `ScrollBarType`.)

`activeRegion`
: The region of the scroll bar that is receiving the pen down events. Possible values are:

  `sclUpArrow`
  : The up arrow.

  `sclDownArrow`
  : The down arrow.

  `sclUpPage`
  : The region between the scroll car and the up arrow.

  `sclDownPage`
  : The region between the scroll car and the down arrow.

  `sclCar`
  : The scroll car.

ScrollBarPtr Typedef

Purpose The `ScrollBarPtr` type defines a pointer to a `ScrollBarType` structure.

Prototype

```
typedef ScrollBarType *ScrollBarPtr;
```

You pass the `ScrollBarPtr` as an argument to all scroll bar functions. You can obtain the `ScrollBarPtr` using the function `FrmGetObjectPtr()` in this way:

```
scrollBarPtr = FrmGetObjectPtr(frm,
  FrmGetObjectIndex(frm, scrollBarID));
```

where `scrollBarID` is the resource ID assigned when you created the scroll bar.

ScrollBarType Struct

Purpose The `ScrollBarType` represents a scroll bar.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `ScrollBarType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  RectangleType bounds;
  UInt16 id;
  ScrollBarAttrType attr;
  Int16 value;
  Int16 minValue;
  Int16 maxValue;
  Int16 pageSize;
  Int16 penPosInCar;
  Int16 savePos;
} ScrollBarType;
```

Your code should treat the `ScrollBarType` structure as opaque. Use the functions described in this chapter to retrieve and set each value. Do not attempt to change structure member values directly.

Fields

`bounds`

Position (using absolute coordinates) and size (in pixels) of the scroll bar on the screen. For example, to access the bounds of an object in a form whose ID is `kObjectID`:

```
{
RectangleType rect;
FormPtr formP = FrmGetActiveForm();


FrmGetObjectBounds(formP,
FrmGetObjectIndex(formP, kObjectID),
&rect);
}
```

`id`
: ID value you specified when you created the scroll bar object.

`attr`
: Scroll bar's attributes. See `ScrollBarAttrType`.

`value`
: Current value of the scroll bar. This value is used to determine where to position the scroll car (the dark region in the scroll bar that indicates the position in the document). Access with `SclGetScrollBar()`.

  The number given is typically a number relative to `minValue` and `maxValue`. These values have nothing to do with any physical characteristics of the object that the scroll bar is attached to, such as the number of lines in the object.

  This value is typically set to 0 initially and then adjusted programmatically with `SclSetScrollBar()`.

`minValue`
: Minimum value. When `value` equals `minValue`, the scroll car is positioned at the very top of the scrolling region. This value is typically 0. Access with `SclGetScrollBar()`.

`maxValue`
: Maximum value. When `value` equals `maxValue`, the scroll car is positioned at the very bottom of the scrolling region. This value is typically set to 0 initially and then adjusted programmatically with `SclSetScrollBar()`. Access with `SclGetScrollBar()`.

`pageSize`
: Number of lines to scroll when user scrolls one page. Access with `SclGetScrollBar()`.

`penPosInChar`
: Used internally.

`savePos`
Used internally.

Scroll Bar Resources

The Scroll Bar Resource (tSCL) represents a scroll bar. See *Palm OS User Interface Guidelines* for more information.

Scroll Bar Functions

SclDrawScrollBar Function

Purpose Draw a scroll bar.

Declared In `ScrollBar.h`

Prototype
```
void SclDrawScrollBar (ScrollBarType *bar)
```

Parameters → *bar*
Pointer to a scroll bar structure (see `ScrollBarType`).

Returns Returns nothing.

Comments This function is called internally by `SclSetScrollBar()` and `FrmDrawForm()`. You rarely need to call it yourself.

Compatibility Implemented only if 2.0 New Feature Set is present.

SclGetScrollBar Function

Purpose Retrieve a scroll bar's current position, its range, and the size of a page.

Declared In `ScrollBar.h`

Prototype
```
void SclGetScrollBar (const ScrollBarType *bar,
    Int16 *valueP, Int16 *minP, Int16 *maxP,
    Int16 *pageSizeP)
```

Parameters → *bar*
Pointer to a scroll bar structure (see `ScrollBarType`).

← *valueP*
A value representing the scroll car's current position. (The scroll car is the dark region that indicates the position in the document.)

← *minP*
A value representing the top of the user interface object.

← *maxP*
A value representing the bottom of the user interface object.

← *pageSizeP*
Pointer to size of a page (used when page scrolling).

Returns Returns the scroll bar's current values in `valueP`, `minP`, `maxP`, and `pageSizeP`.

Comments You might use this function immediately before calling `SclSetScrollBar()` to update the scroll bar. `SclGetScrollBar` returns the scroll bar's current values, which you can then adjust as necessary and pass to `SclSetScrollBar`.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also SclSetScrollBar()

SclHandleEvent Function

Purpose Handles events that affect a scroll bar.

Declared In `ScrollBar.h`

Prototype

```
Boolean SclHandleEvent (ScrollBarType *bar,
    const EventType *event)
```

Parameters

→ *bar*
Pointer to a scroll bar structure (see `ScrollBarType`).

→ *event*
Pointer to an event (EventType).

Returns Returns `true` if the event was handled.

Comments When a `penDownEvent` occurs, the scroll bar sends an `sclEnterEvent` to the event queue.

When an `sclEnterEvent` occurs, the scroll bar determines what its new value should be based on which region of the scroll bar is

receiving the pen down events. It then sends either an `sclRepeatEvent` or an `sclExitEvent` to the event queue.

When the user holds and drags the scroll bar with the pen, the scroll bar sends a `sclRepeatEvent`. Applications that implement dynamic scrolling should catch this event and move the text each time one arrives.

When the user releases the pen from the scroll bar, the scroll bar sends a `sclExitEvent`. Applications that implement non-dynamic scrolling should catch this event and move the text when `sclExitEvent` arrives. Applications that implement dynamic scrolling can ignore this event.

Compatibility Implemented only if 2.0 New Feature Set is present.

SclSetScrollBar Function

Purpose Set the scroll bar's current position, its range, and the size of a page. If the scroll bar is visible and its minimum and maximum values are not equal, it's redrawn.

Declared In `ScrollBar.h`

Prototype

```
void SclSetScrollBar (ScrollBarType *bar,
    Int16 value, Int16 min, Int16 max,
    Int16 pageSize)
```

Parameters

→ *bar*
Pointer to a scroll bar structure (see `ScrollBarType`).

→ *value*
The position the scroll car should move to. (The scroll car is the dark region that indicates the position in the document.)

→ *min*
Minimum value.

→ *max*
Maximum value.

→ *pageSize*
Number of lines of text that can be displayed on a the screen at one time (used when page scrolling).

Returns Returns nothing. May display a fatal error message if the `min` parameter is greater than the `max` parameter.

Comments Call this function when the user adds or deletes text in a field or when a table row is added or deleted.

For scrolling fields, your application should catch the `fldChangedEvent` and update the scroll bar at that time.

The `max` parameter is computed as:

number of lines of text – page size + overlap

where number of lines of text is the total number of lines or rows needed to display the entire object, page size is the number of lines or rows that can be displayed on the screen at one time, and overlap is the number of lines or rows from the bottom of one page to be visible at the top of the next page.

For example, if you have 100 lines of text and 10 lines show on a page, the `max` value would be 90 or 91, depending on the overlap. So if value is greater than or equal to 90 or 91, the scroll car is at the very bottom of the scrolling region.

You can use the `FldGetScrollValues()` function to compute the values you pass for `value`, `min`, and `max`. For example:

```
FldGetScrollValues (fld, &scrollPos,
  &textHeight,  &fieldHeight);

if (textHeight > fieldHeight)
  maxValue = textHeight - fieldHeight;
else if (scrollPos)
  maxValue = scrollPos;
else
  maxValue = 0;

SclSetScrollBar (bar, scrollPos, 0, maxValue,
  fieldHeight-1);
```

In this case, `textHeight` is the number of lines of text and `fieldHeight` is the page size. No lines overlap when you scroll one page. Notice that if the page size is greater than the lines of text, then `max` equals `min`, which means that the scroll bar is not displayed.

For scrolling tables, there is no equivalent to `FldGetScrollValues`. Your application must scroll the table itself and keep track of the scroll values. See the `ListViewUpdateScrollers` function in the Memo example application (`MemoMain.c`) for an example of setting scroll bar values for a table.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `SclGetScrollBar()`

19

System Dialogs

This chapter provides reference material for system dialogs declared in the header files `FatalAlert.h`, `Launcher.h`, `GraffitiReference.h`, and `GraffitiUI.h`.

System Dialog Functions

SysAppLauncherDialog Function

Purpose Display the launcher popup, get a choice, ask the system to launch the selected application, clean up, and leave. If there are no applications to launch, nothing happens.

Declared In `Launcher.h`

Prototype

```
void SysAppLauncherDialog()
```

Parameters `None.`

Returns The system may be asked to launch an application.

Comments Typically, this routine is called by the system as necessary. Most applications do not need to call this function themselves.

In Palm OS® version 3.0 and higher the launcher is an application, rather than a popup. This function remains available for compatibility purposes only.

See Also `SysAppLaunch()`, the "Application Launcher" section in the *Palm OS Programmer's Companion*, vol. I

SysFatalAlert Function

Purpose Display a fatal alert until the user taps a button in the alert.

Declared In `FatalAlert.h`

Prototype

```
UInt16 SysFatalAlert (const Char *msg)
```

Parameters

`msg`
Message to display in the dialog.

Returns The button tapped; first button is zero.

SysGraffitiReferenceDialog Function

Purpose Pop up the Graffiti® or Graffiti 2 Reference Dialog.

Declared In `GraffitiReference.h`

Prototype

```
void SysGraffitiReferenceDialog
    (ReferenceType referenceType)
```

Parameters

`referenceType`
Which reference to display. See `GraffitiReference.h` for more information.

Returns Nothing returned.

20

Tables

This chapter describes the table API as declared in the header file `Table.h`. It discusses the following topics:

- Table Data Structures
- Table Constants
- Table Resource
- Table Functions
- Application-Defined Functions

For more information on tables, see the section "Tables" in the *Palm OS Programmer's Companion*, vol. I.

Table Data Structures

TableAttrType Struct

Purpose The `TableAttrType` bit field defines the visible characteristics of the table.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `TableAttrType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  UInt16 visible:1;
  UInt16 editable:1;
  UInt16 editing:1;
  UInt16 selected:1;
  UInt16 hasScrollBar:1;
  UInt16 reserved:11;
} TableAttrType;
```

Your code should treat the `TableAttrType` bit field as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change member values directly.

Fields

`visible`
: If set, the table is drawn on the screen. The value of this field is set by `TblDrawTable()` and cleared by `TblEraseTable()`.

`editable`
: If set, the user can modify the table. You specify this when you create the table resource.

`editing`
: If set, the table is in edit mode. The table is in edit mode while the user edits a text item. The value of this field is returned by `TblEditing()`.

`selected`
: If set, the current item (as identified by the `TableType` fields `currentRow` and `currentColumn`) is selected. Use `TblGetSelection()` to retrieve this value.

`hasScrollBar`
: If set, the table has a scroll bar. Note that this attribute can only be set programmatically. See `TblHasScrollBar()`.

TableColumnAttrType Struct

Purpose

The `TableColumnAttrType` structure defines a column in a table.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `TableColumnAttrType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  Coord width;
  UInt16 reserved1 : 5;
  UInt16 masked : 1;
  UInt16 editIndicator : 1;
  UInt16 usable : 1;
  UInt16 reserved2 : 8;
  Coord spacing;
  TableDrawItemFuncPtr drawCallback;
  TableLoadDataFuncPtr loadDataCallback;
  TableSaveDataFuncPtr saveDataCallback;
} TableColumnAttrType;
```

Your code should treat the `TableColumnAttrType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

Fields

`width`
: The column's width in pixels. See `TblGetColumnWidth()` and `TblSetColumnWidth()`.

`reserved1`
: Reserved for future use.

`masked`
: If `true` and the item's row also has a `masked` attribute of `true`, the table cell is drawn on the screen but is shaded to obscure the information that it contains. See `TblSetColumnMasked()`.

`editIndicator`
: If `true`, items in the column should be highlighted if selected while in edit mode. If `false`, items in the column should not be highlighted. By default, text field items are highlighted in edit mode, but all other types of items are not highlighted. The default can be overridden with `TblSetColumnEditIndicator()`.

`usable`
: If `false`, the column is not considered part of the current interface of the application, and it doesn't appear on screen. See `TblSetColumnUsable()`.

`reserved2`
: Reserved for future use.

`spacing`
: The spacing in pixels between this column and the next column. See `TblGetColumnSpacing()` and `TblSetColumnSpacing()`.

`drawCallback`
: Pointer to a function that draws custom items in the column. This function is called during `TblDrawTable()` and `TblRedrawTable()`. See `TblSetCustomDrawProcedure()`.

`loadDataCallback`
: Pointer to a function that loads data into the column. This function is called during `TblDrawTable()` and `TblRedrawTable()`. See `TblSetLoadDataProcedure()`.

`saveDataCallback`
: Pointer to a function that saves the data in the column. Called when the focus moves from one table cell to another and when the table loses focus entirely. See `TblSetSaveDataProcedure()`.

Compatibility The `masked` field is defined only if 3.5 New Feature Set is present.

TableItemPtr Typedef

Purpose A `TableItemPtr` points to a `TableItemType`.

Prototype

```
typedef TableItemType *TableItemPtr;
```

TableItemType Struct

Purpose The `TableItemType` structure defines an item, or cell, within the table.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `TableItemType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  TableItemStyleType itemType;
  FontID fontID;
  Int16 intValue;
  Char *ptr;
} TableItemType;
```

Your code should treat the `TableItemType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

NOTE: None of the table items create memory that you need to free. The table manager handles all of the allocating and deallocating of memory for table items. The only memory you are responsible for freeing is the memory handle containing the text that you want displayed in editable text fields. (See `TableLoadDataFuncType()`.)

Fields

`itemType`
: The type of the item, such as a control, a text label, and so on. `TblSetItemStyle()` sets this value. The rest of the fields in this struct are either used or not used depending on the `itemType`. See Table 20.1.

`fontID`
: ID of the font used to display the item's text. `TblGetItemFont()` and `TblSetItemFont()` retrieve and set this value.

`intValue`
: Integer value of the item. `TblGetItemInt()` and `TblSetItemInt()` retrieve and set this value.

`ptr`

Pointer to the item's text. `TblGetItemPtr()` and `TblSetItemPtr()` retrieve and set this value.

All text items have a maximum of `tableMaxTextItemSize`.

The following table lists the possible values for the `itemType` field, describes how each type is drawn, describes which other fields are used for each `itemType`, and provides special instructions for setting those fields. Note in particular that the `fontID` field is often not used. Instead, certain items are displayed in a standard font. These are noted in the last column of this table.

Table 20.1 TableItemType fields

ItemType	Description	TableItemType Fields Used
`checkboxTableItem`	A checkbox control.	`intValue`
`customTableItem`	Application-defined cell. The height of the item is fixed at 11 pixels.	None. Custom items are drawn using the custom drawing function that you implement. See `TableDrawItemFuncType()`. If you want, you can store data in the `intValue` and `ptr` fields.

Table 20.1 TableItemType fields *(continued)*

ItemType	Description	TableItemType Fields Used

`dateTableItem`	Non-editable date in the form *month/day*, or a dash if the date value is -1. The date is followed by an exclamation point if it has past.	`intValue`
		The `intValue` field should be a value that can be cast to `DateType`. `DateType` is currently defined as a 16-bit number:
		`yyyyyyymmmmddddd`
		The first 7 bits are the year given as the offset since 1904, the next 4 bits are the month, and the next 5 bits are the day.
		Dates are always drawn in the current font.

`labelTableItem`	Non-editable text.	`ptr` Labels are displayed in the system's default font along with a terminating colon character (':'). Use a `customTableItem` or `tallCustomTableItem` if you don't want a colon.
`numericTableItem`	Non-editable number.	`intValue` Numbers are displayed in the system's default bold font.

Table 20.1 TableItemType fields *(continued)*

ItemType	Description	TableItemType Fields Used

`popupTriggerTableItem`	A list with a pop-up trigger.	`intValue`
		`ptr`
		`intValue` is the index of the list item that should be displayed.
		`ptr` is a pointer to the list.
		Lists are displayed in the system's default font.

`tallCustomTableItem`	Application-defined cell. The height of the item is equal to the height of the row in which the item is located. This table item type was added in Palm OS 4.0 and is only supported if 4.0 New Feature Set is defined.	None. Custom items are drawn using the custom drawing function that you implement. See TableDrawItemFuncType(). If you want, you can store data in the `intValue` and `ptr` fields.

`textTableItem`	Editable text field.	`fontID`
		`ptr`
		For this item type, implement the callback function `TableLoadDataFuncType()` to load text into the table cell and implement the callback `TableSaveDataFuncType()` to save data before the field is freed.

Table 20.1 TableItemType fields *(continued)*

ItemType	Description	TableItemType Fields Used

`textWithNoteTableItem`	Editable text field and a note icon to the right of the text.	`fontID`
		`ptr`
		For this item type, implement the callback function `TableLoadDataFuncType()` to load text into the table cell and implement the callback `TableSaveDataFuncType()` to save data before the field is freed.

`timeTableItem`	Not implemented.	

`narrowTextTableItem`	Editable text with space reserved on the right side of the cell.	`fontID`
		`ptr`
		`intValue`
		`intValue` is the number of pixels to reserve on the right side of the cell.
		For this item type, implement the callback function `TableDrawItemFuncType()` to draw in the space reserved on the right side of the cell, the `TableLoadDataFuncType()` callback function to load text into the table cell, and the callback function `TableSaveDataFuncType()` to save data before the field is freed.

TablePtr Typedef

Purpose The `TablePtr` type defines a pointer to a `TableType`.

Prototype

```
typedef TableType *TablePtr;
```

You pass the table's pointer as an argument to all table functions. You can obtain the table's pointer using the function `FrmGetObjectPtr()` in this way:

```
tblPtr = FrmGetObjectPtr(frm,
  FrmGetObjectIndex(frm, tblID));
```

where `tblID` is the resource ID assigned when you created the table.

TableRowAttrType Struct

Purpose The `TableRowAttrType` structure defines a row in a table.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `TableRowAttrType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct {
  UInt16 id;
  Coord height;
  UInt32 data;
  UInt16 reserved1 : 7;
  UInt16 usable : 1;
  UInt16 reserved2 : 4;
  UInt16 masked : 1;
  UInt16 invalid : 1;
  UInt16 staticHeight : 1;
  UInt16 selectable : 1;
  UInt16 reserved3;
} TableRowAttrType;
```

Your code should treat the `TableRowAttrType` structure as opaque. Use the functions specified in the descriptions below to

retrieve and set each value. Do not attempt to change structure member values directly.

Fields

`id`
: The ID of this row. See `TblFindRowID()`, `TblGetRowID()`, and `TblSetRowID()`.

`height`
: Height of the row in pixels. The functions `TblSetRowHeight()` and `TblGetRowHeight()` set and retrieve this value.

`data`
: Any application-specific value you want to store in this row. For example, the Datebook and ToDo applications use this field to store the unique ID of the database record that is displayed in this table row. See `TblFindRowData()`, `TblGetRowData()`, and `TblSetRowData()`.

`reserved1`
: Reserved for future use.

`usable`
: If `false`, the row is not considered part of the current interface of the application, and it doesn't appear on screen. Table rows have `usable` set to `false` when they are scrolled off the screen. See `TblRowUsable()` and `TblSetRowUsable()`. The function `TblGetLastUsableRow()` returns the row that appears at the bottom of the screen.

`masked`
: If `true` and the item's column also has a `masked` attribute of `true`, the table cell is drawn on the screen but is shaded to obscure the information that it contains. See `TblSetRowMasked()` and `TblRowMasked()`.

`reserved2`
: Reserved for future use.

`invalid`
: If `true`, the row needs to be redrawn. See `TblRowInvalid()`, `TblMarkRowInvalid()`, and `TblMarkTableInvalid()`.

`staticHeight`
: `true` if the row height cannot be changed, `false` otherwise. If `false`, text fields in this table row will dynamically resize to multiple lines as necessary. See `TblSetRowStaticHeight()`.

`selectable`
: If `true`, the user can select individual cells in this row. See `TblSetRowSelectable()` and `TblRowSelectable()`.

`reserved3`
: Reserved for future use.

Compatibility The `masked` field is defined only if 3.5 New Feature Set is present.

TableType Struct

Purpose The `TableType` structure represents a table.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `TableType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct TableType {
  UInt16 id;
  RectangleType bounds;
  TableAttrType attr;
  Int16 numColumns;
  Int16 numRows;
  Int16 currentRow;
  Int16 currentColumn;
  Int16 topRow;
  TableColumnAttrType *columnAttrs;
  TableRowAttrType *rowAttrs;
  TableItemPtr items;
  FieldType currentField;
} TableType;
```

Your code should treat the `TableType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

Fields

`id`
ID value you specified when you created the table resource. This ID is included as part of the event data of `tblEnterEvent`.

`bounds`
Position and size of the table object. The functions `TblGetBounds()`, `FrmGetObjectBounds()`, `TblSetBounds()`, and `FrmSetObjectBounds()` retrieve and set this value.

`attr`
The table's attributes. See `TableAttrType`.

`numColumns`
Number of columns displayed by the table object. You specify the number of columns when you create the table resource, and this value cannot be changed. Access with `TblGetNumberOfColumns()`.

`numRows`
Maximum number of visible rows in the table object.

You specify this value when you create the table resource, and it does not change; however, the total number of rows in a table can change if you insert new rows in a table, and even the number of currently visible rows can change if a text field within a table cell is dynamically resized.

The function `TblGetNumberOfRows()` returns the value of this field.

`currentRow`
Row index of the currently selected table cell. Rows are numbered from top to bottom starting with 0. `TblGetSelection()` and `TblSetSelection()` retrieve and set the values of `currentRow`.

`currentColumn`
Column index of the currently selected table cell. Columns are numbered from left to right starting with 0. If the `TableAttrType` `selected` is `true`, then this table cell is highlighted. If `selected` is `false`, the table still considers this the "current" item, but it is not highlighted. `TblGetSelection()` and `TblSetSelection()` retrieve and set the values of `currentColumn`.

`topRow`
: First visible row of the table object. Access with `TblGetTopRow()`.

`columnAttrs`
: An array of each table column's attributes. See `TableColumnAttrType`.

`rowAttrs`
: An array of each row's attributes, such as its ID, height, and whether or not it is usable, selectable, or invalid. See `TableRowAttrType`.

`items`
: An array of each item's (table cell's) attributes, such as the item type, font ID, an integer value, and a character pointer. See `TableItemType`.

`currentField`
: Field object the user is currently editing. The function `TblGetCurrentField()` retrieves the value of this item.

Table Constants

Constant	Value	Description
`tableDefaultColumnSpacing`	1	Never used.
`tableNoteIndicatorHeight`	11	The height in pixels of the note indicator for tables items of type `textWithNoteTableItem`.
`tableNoteIndicatorWidth`	7	The width in pixels of the note indicator for tables items of type `textWithNoteTableItem`.
`tableMaxTextItemSize`	255	The maximum length of an editable text field within a table cell.
`tblUnusableRow`	0xffff	Value returned by `TblGetLastUsableRow()` if none of the table's rows are usable. This value is only available in version 3.5 and higher.

Table Resource

The Table Resource (tTBL) represents a table on screen. See *Palm OS User Interface Guidelines* for more information.

Table Functions

TblDrawTable Function

Purpose Draw a table.

Declared In `Table.h`

Prototype

```
void TblDrawTable (TableType *tableP)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

Returns Returns nothing.

Comments This function is called as part of `FrmDrawForm()` when the form contains a table object.

This function draws the entire table, marking all rows valid before drawing. See the `TableItemType` struct description for more information about how each type of table cell is drawn.

When drawing cells with editable text fields (`textTableItem`, `textWithNoteTableItem`, or `narrowTextTableItem`), this function uses the `TableLoadDataFuncType()` callback function to load the text into the table cells. The text field does not retain the text handle that your `TableLoadDataFunc` returns, meaning that you are responsible for freeing the memory that you load into the table.

When drawing `narrowTextTableItem` cells, `customTableItem` cells or `tallCustomTableItem` cells, this function uses the `TableDrawItemFuncType()` callback function to draw the extra pixels after the text or to draw the entire cell.

On color systems, tables are always drawn using the same color as is used for the field background color.

When the user has set the security setting to mask private records, table cells that contain private database records are drawn as shaded cells to obscure the information they contain. You must explicitly tell the table which cells are masked by making the appropriate calls to `TblSetRowMasked()` and `TblSetColumnMasked()`.

Compatibility Color support and masked private records are only supported in Palm OS® version 3.5 and higher.

In versions earlier than 3.5, this function did not erase table cells before it drew them. In earlier releases, consider calling `TblEraseTable()` before calling this function, particularly if the entire table has changed, as the visual effect of drawing over a white background may be more pleasing.

See Also `TblEraseTable()`, `TblRedrawTable()`, `TblSetCustomDrawProcedure()`

TblEditing Function

Purpose Check whether a table is in edit mode.

Declared In `Table.h`

Prototype

```
Boolean TblEditing (const TableType *tableP)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

Returns Returns `true` if the table is in edit mode, `false` otherwise.

Comments The table is in edit mode while the user edits a text item. More specifically, the table is in edit mode when a `tblEnterEvent` is received on an editable table cell (`textTableItem`, `textWithNoteTableItem`, or `narrowTextTableItem`), or when `TblGrabFocus()` is called.

The table is taken out of edit mode when a the user places the pen on a note in a `textWithNoteTableItem` or when the table releases the focus (`TblReleaseFocus()`).

TblEraseTable Function

Purpose Erase a table object.

Declared In `Table.h`

Prototype

```
void TblEraseTable (TableType *tableP)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

Returns Returns nothing.

Comments This function sets the table's `visible` and `selected` attributes to `false`. It does not invalidate table rows.

See Also `TblDrawTable()`, `TblSetCustomDrawProcedure()`, `TblRedrawTable()`

TblFindRowData Function

Purpose Return the number of the row that contains the specified data value.

Declared In `Table.h`

Prototype

```
Boolean TblFindRowData (const TableType *tableP,
   UInt32 data, Int16 *rowP)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *data*
Row data to find.

← *rowP*
Pointer to the row number (return value).

Returns Returns `true` if a match was found, `false` otherwise.

Comments This function searches for a row whose attributes contain the specified data. The data is any application-specific data that you have set with `TblSetRowData()`.

See Also `TblGetRowData()`, `TblFindRowID()`, `TableRowAttrType`

TblFindRowID Function

Purpose Return the number of the row with the specified ID.

Declared In `Table.h`

Prototype

```
Boolean TblFindRowID (const TableType *tableP,
    UInt16 id, Int16 *rowP)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *id*
Row ID to find.

← *rowP*
Pointer to the row number (return value).

Returns Returns `true` if a match was found, `false` otherwise.

See Also `TblSetRowID()`, `TblFindRowData()`, `TableRowAttrType`

TblGetBounds Function

Purpose Return the bounds of a table.

Declared In `Table.h`

Prototype

```
void TblGetBounds (const TableType *tableP,
    RectangleType *rP)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

← *rP*
A `RectangleType` structure in which the bounds are returned.

Returns Returns nothing. The `rP` parameter contains the bounds.

See Also `TblGetItemBounds()`

TblGetColumnSpacing Function

Purpose Return the spacing after the specified column.

Declared In `Table.h`

Prototype

```
Coord TblGetColumnSpacing
    (const TableType *tableP, Int16 column)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *column*
Column number (zero-based).

Returns Returns the spacing after column (in pixels).

This function may display a fatal error message if the `column` parameter is invalid.

See Also `TblGetColumnWidth()`, `TblSetColumnSpacing()`, `TblSetColumnUsable()`

TblGetColumnWidth Function

Purpose Return the width of the specified column.

Declared In `Table.h`

Prototype

```
Coord TblGetColumnWidth (const TableType *tableP,
    Int16 column)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *column*
Column number (zero-based).

Returns Returns the width of a column (in pixels). This function may display a fatal error message if the `column` parameter is invalid.

See Also `TblGetColumnSpacing()`, `TblSetColumnWidth()`, `TblSetColumnUsable()`

TblGetCurrentField Function

Purpose Return a pointer to the `FieldType` in which the user is currently editing a text item.

Declared In `Table.h`

Prototype

```
FieldPtr TblGetCurrentField
    (const TableType *tableP)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

Returns Returns a pointer to the currently selected field, or `NULL` if the table is not in edit mode.

See Also TblGetSelection()

TblGetItemBounds Function

Purpose Return the bounds of an item in a table.

Declared In `Table.h`

Prototype

```
void TblGetItemBounds (const TableType *tableP,
    Int16 row, Int16 column, RectangleType *rP)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number of the item (zero-based).

→ *column*
Column number of the item (zero-based).

← *rP*
Pointer to a structure that holds the bounds of the item.

Returns Returns nothing. Stores the bounds in `rP`. This function may raise a fatal exception if the `row` or `column` parameter specifies a row or column that does not appear on screen.

TblGetItemFont Function

Purpose Return the font used to display a table item.

Declared In `Table.h`

Prototype

```
FontID TblGetItemFont (const TableType *tableP,
    Int16 row, Int16 column)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number of the item (zero-based).

→ *column*
Column number of the item (zero-based).

Returns Returns the ID of the font used for the table item at the row and column indicated. This function may display a fatal error message if the `row` or `column` parameter specifies a row or column that is not on the screen.

Comments This function returns the value stored in the `fontID` field for this table item. Only certain types of table items use the font specified by the `fontID` field when they are displayed. The `TableItemType` description specifies what font is used to display each type of table item.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `TblSetItemFont()`

TblGetItemInt Function

Purpose Return the integer value stored in a table item.

Declared In `Table.h`

Prototype

```
Int16 TblGetItemInt (const TableType *tableP,
    Int16 row, Int16 column)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number of the item (zero-based).

→ *column*
Column number of the item (zero-based).

Returns Returns the integer value. This function may display a fatal message if the `row` or `column` does not appear on the screen.

Comments This function returns the value stored in the `intValue` field for this table item. Certain types of table items display the value stored in `intValue`, and other types display the value pointed to by the `ptr` field. See the `TableItemType` description for details. If the `intValue` was never set for this table item, this function returns 0.

See Also `TblSetItemInt()`, `TblGetItemPtr()`

TblGetItemPtr Function

Purpose Return the pointer value stored in a table item

Declared In `Table.h`

Prototype

```
void *TblGetItemPtr (const TableType *tableP,
   Int16 row, Int16 column)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number of the item (zero-based).

→ *column*
Column number of the item (zero-based).

Returns Returns the item's pointer value or `NULL` if the item does not have a pointer value. This function may display a fatal message if the `row` or `column` parameter is invalid.

Comments This function returns the value stored in the `ptr` field for this table item. Certain types of table items display the value pointed to by the `ptr`, and other types display the value stored in the `intValue` field. See the `TableItemType` description for details. An application may have set the value of the `ptr` field anyway, even for items that use the `intValue`. This function always returns that value.

Compatibility Implemented only if 3.5 New Feature Set is present. In earlier versions, you can implement this function using the following code:

```
return tableP->items[row * tableP->numColumns +
  column].ptr;
```

See Also TblSetItemPtr()

TblGetLastUsableRow Function

Purpose Return the last row in a table that is usable (visible).

Declared In `Table.h`

Prototype

```
Int16 TblGetLastUsableRow
    (const TableType *tableP)
```

Parameters → *tableP*
Pointer to a table object. (See TableType.)

Returns Returns the row index (zero-based) or `tblUnusableRow` if there are no usable rows.

See Also TblGetRowData(), TblGetRowID()

TblGetNumberOfColumns Function

Purpose Return the number of columns in a table.

Declared In `Table.h`

Prototype

```
Int16 TblGetNumberOfColumns
    (const TableType *tableP)
```

Parameters → *tableP*
Pointer to a TableType.

Returns This function returns the number of columns in a table.

Compatibility Implemented only if 4.0 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TblGlueGetNumberOfColumns`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also TblGetTopRow(), TblSetSelection()

TblGetNumberOfRows Function

Purpose Return the number of rows in a table.

Declared In `Table.h`

Prototype

```
Int16 TblGetNumberOfRows
    (const TableType *tableP)
```

Parameters → *tableP*
Pointer to a table object. (See `TableType`.)

Returns Returns the maximum number of visible rows in the specified table.

Comments Note that even though you can add and remove rows to and from a table, the value returned by this function does not change. The value returned by this function indicates the maximum number of rows that can be displayed on the screen at one time. It is set when you create the table resource.

TblGetRowData Function

Purpose Return the data value of the specified row.

Declared In `Table.h`

Prototype

```
UInt32 TblGetRowData (const TableType *tableP,
    Int16 row)
```

Parameters → *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Number of the row (zero-based).

Returns Returns the application-specific data stored for this row, if any. Returns 0 if there is no application-specific data value.

This function may display a fatal error message if the `row` parameter is invalid.

See Also `TblFindRowData()`, `TblSetRowData()`, `TableRowAttrType`

TblGetRowHeight Function

Purpose Return the height of the specified row.

Declared In `Table.h`

Prototype

```
Coord TblGetRowHeight (const TableType *tableP,
    Int16 row)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Number of the row (zero-based).

Returns Returns the height in pixels. This function may display a fatal error message if the `row` parameter is invalid.

See Also `TblGetItemBounds()`, `TblSetRowHeight()`

TblGetRowID Function

Purpose Return the ID value of the specified row.

Declared In `Table.h`

Prototype

```
UInt16 TblGetRowID (const TableType *tableP,
    Int16 row)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Number of the row (zero-based).

Returns Returns the ID value of the row in the table.

This function may display a fatal error message if the `row` parameter is invalid.

See Also `TblGetRowData()`, `TblSetRowID()`, `TblFindRowID()`, `TableRowAttrType`

TblGetSelection Function

Purpose Return the row and column of the currently selected table item.

Declared In `Table.h`

Prototype

```
Boolean TblGetSelection (const TableType *tableP,
    Int16 *rowP, Int16 *columnP)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

← *rowP, columnP*
The row and column indexes (zero-based) of the currently selected item.

Returns Returns `true` if the item is highlighted, `false` if not.

See Also `TblSetRowSelectable()`

TblGetTopRow Function

Purpose Return the top row visible row of a table.

Declared In `Table.h`

Prototype

```
Int16 TblGetTopRow(const TableType *tableP)
```

Parameters

→ *tableP*
Pointer to a `TableType`.

Returns `This function returns the top visible row in a table.`

Compatibility Implemented only if 4.0 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TblGlueGetToRow`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `TblGetNumberOfColumns()`, `TblSetSelection()`

TblGrabFocus Function

Purpose Put a table into edit mode.

Declared In `Table.h`

Prototype

```
void TblGrabFocus (TableType *tableP, Int16 row,
    Int16 column)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Current row to be edited (zero-based).

→ *column*
Current column to be edited (zero-based).

Returns Returns nothing. This function may display a fatal error message if the table already has the focus or if the `row` or `column` parameter is invalid.

Comments This function puts the table into edit mode and sets the current item to the one at the row and column passed in. An editable field must exist in the coordinates passed to this function.

You must call `FrmSetFocus()` before calling this function. `FrmSetFocus` releases the focus from the object that previously had it and sets the form's internal structures. After calling this function, you must call `FldGrabFocus()` to display the insertion point in the field. (You can use `TblGetCurrentField()` to obtain a pointer to the field.)

For example, the following function from the Address Book application sets the focus in an editable field within a table:

```
static void EditViewRestoreEditState () {
  Int16      row;
  FormPtr    frm;
  TablePtr   table;
  FieldPtr   fld;

  if (CurrentFieldIndex == noFieldIndex)
    return;

  // Find the row that the current field is in.
  table = GetObjectPtr (EditTable);
  if ( ! TblFindRowID (table,
```

```
    CurrentFieldIndex, &row) )
    return;

  frm = FrmGetActiveForm ();
  FrmSetFocus (frm, FrmGetObjectIndex (frm,
    EditTable));
  TblGrabFocus (table, row, editDataColumn);

  // Restore the insertion point position.
  fld = TblGetCurrentField (table);
  FldSetInsPtPosition (fld, EditFieldPosition);
  FldGrabFocus (fld);
}
```

See Also TblReleaseFocus()

TblHandleEvent Function

Purpose Handle an event for the table.

Declared In `Table.h`

Prototype

```
Boolean TblHandleEvent (TableType *tableP,
    EventType *event)
```

Parameters

→ *tableP*
Pointer to a table object. (See TableType.)

→ *event*
The event to be handled.

Returns Returns `true` if the event was handled, `false` if it was not.

Comments Returns `false` if the table is not an editable table.

If the table is editable, this function passes along any keyDownEvent, fldEnterEvent, or menuCmdBarOpenEvent to the currently selected field.

When a fldHeightChangedEvent occurs, this function changes the height of the specified field as indicated by the event. If the field being resized is going to scroll off the bottom of the screen, then instead the table scrolls the rows above it up off the top. Otherwise, the table is scrolled downward and rows below the current row are scrolled off the bottom as necessary.

Note that the `fldHeightChangedEvent` is only handled for dynamically sized fields. See the descriptions of `FieldAttrType` and `FldMakeFullyVisible()` for more information.

When a `penDownEvent` occurs, the table checks to see if the focus is being changed. If it is and the user was previously editing a text field within the table, it saves the data in the table cell using the `TableSaveDataFuncType()` callback function, then it enqueues a `tblEnterEvent` with the new row and column that are selected.

When a `tblEnterEvent` occurs, this function tracks the pen until it is lifted. If the pen is lifted within the bounds of the same item it went down in, a `tblSelectEvent` is added to the event queue; if not, a `tblExitEvent` is added to the event queue.

TblHasScrollBar Function

Purpose Set the `hasScrollBar` attribute in the table. (See `TableAttrType`.)

Declared In `Table.h`

Prototype

```
void TblHasScrollBar (TableType *tableP,
    Boolean hasScrollBar)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *hasScrollBar*
`true` to set the attribute, `false` to unset it.

Returns Returns nothing.

Comments Your application must scroll the table itself and keep track of the scroll values. See the `ListViewUpdateScrollers` function in the Memo example application (`MemoMain.c`) for an example of setting scroll bar values for a table.

Compatibility Implemented only if 2.0 New Feature Set is present.

TblInsertRow Function

Purpose Insert a row into the table before the specified row.

Declared In `Table.h`

Prototype
```
void TblInsertRow (TableType *tableP, Int16 row)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row to insert (zero-based).

Returns Returns nothing.

Comments The number of rows in a table is the maximum number of rows displayed on the screen. Unlike a multi-line text field, there is no notion of a table that is bigger than the available screen. For this reason, this function does not increase the number of table rows.

Instead of keeping track of a total number of rows in the table and a number of rows displayed on the screen, tables mark any row that isn't currently displayed with a usable value of `false`. (See `TableRowAttrType`.)

The newly inserted row is marked as invalid, unusable, and not masked. If you want to display the newly inserted row, call `TblSetRowUsable()` after making sure that the row displays a value and then call `TblRedrawTable()` when you are ready to draw the table.

See Also `TblRemoveRow()`, `TblSetRowUsable()`, `TblSetRowSelectable()`

TblMarkRowInvalid Function

Purpose Mark the row invalid.

Declared In `Table.h`

Prototype
```
void TblMarkRowInvalid (TableType *tableP,
    Int16 row)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number (zero-based).

Returns Returns nothing.

Comments After calling this function, call `TblRedrawTable()` to redraw all rows marked invalid.

This function may display a fatal error message if the `row` parameter is invalid.

See Also `TblRemoveRow()`, `TblSetRowUsable()`, `TblSetRowSelectable()`, `TblMarkTableInvalid()`, `TblRowInvalid()`, `TableRowAttrType`

TblMarkTableInvalid Function

Purpose Mark all the rows in a table invalid.

Declared In `Table.h`

Prototype
```
void TblMarkTableInvalid (TableType *tableP)
```

Parameters → *tableP*
Pointer to a table object. (See `TableType`.)

Returns Returns nothing.

Comments After calling this function, you must call `TblRedrawTable()` to redraw all rows.

See Also `TblEraseTable()`, `TblRedrawTable()`, `TableRowAttrType`

TblRedrawTable Function

Purpose Redraw the rows of the table that are marked invalid.

Declared In `Table.h`

Prototype
```
void TblRedrawTable (TableType *tableP)
```

Parameters → *tableP*
Pointer to a table object. (See `TableType`.)

Returns Returns nothing.

Comments This function draws the invalid rows in the table. See the `TableItemType` struct description for more information about how each type of table cell is drawn.

When drawing cells with editable text fields (`textTableItem`, `textWithNoteTableItem`, or `narrowTextTableItem`), this function uses the `TableLoadDataFuncType()` callback function to load the text into the table cells. The text field does not retain the text handle that your `TableLoadDataFunc` returns, meaning that you are responsible for freeing the memory that you load into the table.

When drawing `narrowTextTableItem` cells, `customTableItem` cells, or `tallCustomTableItem` cells, this function uses the `TableDrawItemFuncType()` callback function to draw the extra pixels after the text or to draw the entire cell.

On color systems, tables are always drawn using the same color as is used for the field background color.

When the user has set the security setting to mask private records, table cells that contain private database records are drawn as shaded cells to obscure the information they contain. You must explicitly tell the table which cells are masked by making the appropriate calls to `TblSetRowMasked()` and `TblSetColumnMasked()`.

Compatibility Color support and masked private records are only supported in Palm OS version 3.5 and higher.

See Also `TblMarkTableInvalid()`, `TblMarkRowInvalid()`, `TblDrawTable()`

TblReleaseFocus Function

Purpose Release the focus.

Declared In `Table.h`

Prototype

```
void TblReleaseFocus (TableType *tableP)
```

Parameters → *tableP*
Pointer to a table object.

Returns Returns nothing.

Comments You typically do not call this function yourself. Instead, call `FrmSetFocus()` with an object index of `noFocus` to notify the form that the table has lost focus. The form code calls `TblReleaseFocus` for you.

If the current item is a text item, the `TableSaveDataFuncType()` callback function is called to save the text in the currently selected field, the memory allocated for editing is released, and the insertion point is turned off.

Also note that you might have to call `FldReleaseFocus()` if the focus is in an editable text field and that field uses a custom drawing function (`TableDrawItemFuncType()`).

See Also `TblGrabFocus()`

TblRemoveRow Function

Purpose Remove the specified row from the table.

Declared In `Table.h`

Prototype

```
void TblRemoveRow (TableType *tableP, Int16 row)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row to remove (zero-based).

Returns Returns nothing. This function may raise a fatal error message if an invalid row is specified.

Comments The number of rows in the table is not decreased; instead, this row is moved from its current spot to the end of the table and is marked unusable so that it won't be displayed when the table is redrawn.

This function does not visually update the display. To update the display, call `TblRedrawTable()`.

See Also `TblInsertRow()`, `TblSetRowUsable()`, `TblSetRowSelectable()`, `TblMarkRowInvalid()`

TblRowInvalid Function

Purpose Return whether a row is invalid.

Declared In `Table.h`

Prototype

```
Boolean TblRowInvalid (const TableType *tableP,
    Int16 row)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number (zero-based).

Returns Returns `true` if the row is invalid, `false` if it's valid. This function may raise a fatal error message if the `row` parameter is invalid.

Comments Invalid rows need to be redrawn. Use `TblRedrawTable()` to do so.

See Also `TblMarkRowInvalid()`, `TblMarkTableInvalid()`

TblRowMasked Function

Purpose Return whether a row is masked.

Declared In `Table.h`

Prototype

```
Boolean TblRowMasked (const TableType *tableP,
    Int16 row)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number (zero-based).

Returns Returns `true` if the row is masked, `false` otherwise.

Comments Your code should set a row to masked if it contains a private database record and the user has set the display preference for private records to masked. Masked cells are displayed as shaded.

Note that a table cell is not masked unless both its row and column are masked. This allows non-private information in the row item to remain visible. For example, the Datebook application shows the time for a private appointment, but it does not show the description.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `TblSetRowMasked()`, `TblSetColumnMasked()`, `TableRowAttrType`, `SecSelectViewStatus()`

TblRowSelectable Function

Purpose Return whether the specified row is selectable.

Declared In `Table.h`

Prototype

```
Boolean TblRowSelectable
    (const TableType *tableP, Int16 row)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number (zero-based).

Returns Returns `true` if the row is selectable, `false` if it's not.

Comments Rows that are not selectable don't highlight when touched.

See Also `TableRowAttrType`

TblRowUsable Function

Purpose Determine whether the specified row is usable.

Declared In `Table.h`

Prototype

```
Boolean TblRowUsable (const TableType *tableP,
    Int16 row)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number (zero-based).

Returns Returns `true` if the row is usable, `false` if it's not.

This function may display a fatal error message if the `row` parameter is invalid.

Comments Rows that are not usable do not display.

See Also TblRowSelectable(), TblGetLastUsableRow(), TblSetRowUsable()

TblSelectItem Function

Purpose Select (highlight) the specified item. If there is already a selected item, it is unhighlighted.

Declared In `Table.h`

Prototype

```
void TblSelectItem (TableType *tableP, Int16 row,
    Int16 column)
```

Parameters

→ *tableP*
Pointer to a table object. (See TableType.)

→ *row*
Row of the item to select (zero-based).

→ *column*
Column of the item to select (zero-based).

Returns Returns nothing.

This function may display a fatal error message if the `column` or `row` parameter point to an item that is not on the screen.

Comments If `row` contains a masked private database record, then the item remains unselected.

This function cannot highlight an entire row; it can only highlight one cell in a row, and it always unhighlights the previously selected table cell. If you want to select an entire row, either create a table that has a single column, or write your own selection code.

If the selected item is a multi-line text field or a text field with a nonstandard height, this function only highlights the top eleven pixels. If you want a larger area highlighted, you must write your own selection code.

See Also TblRowSelectable(), TblGetItemBounds(), TblGetItemInt()

TblSetBounds Function

Purpose Sets the bounds of a table.

Declared In `Table.h`

Prototype

```
void TblSetBounds (TableType *tableP,
    const RectangleType *rP)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *rP*
Pointer to a `RectangleType` structure that specifies the bounds for the table.

Returns Returns nothing.

Compatibility Implemented only if 2.0 New Feature Set is present.

TblSetColumnEditIndicator Function

Purpose Set the column attribute that controls whether a column highlights when the table is in edit mode.

Declared In `Table.h`

Prototype

```
void TblSetColumnEditIndicator
    (TableType *tableP, Int16 column,
    Boolean editIndicator)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *column*
Column number (zero based).

→ *editIndicator*
`true` to highlight, `false` to turn off highlight.

Returns Returns nothing.

Comments The edit indicator controls whether the item in the column is highlighted when it is selected. By default, text field items have the `editIndicator` value of `false`, while all other table item types have an edit indicator of `true`.

When the table is drawn, only the leftmost contiguous set of items with the edit indicator set are drawn as highlighted. That is, if

columns 1, 2, and 4 have an edit indicator of `true` and column 3 has an edit indicator of `false`, only the items in column 1 and 2 are drawn as highlighted when selected. Column 4 items are not drawn as highlighted.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `TableColumnAttrType`

TblSetColumnMasked Function

Purpose Set whether the column is masked.

Declared In `Table.h`

Prototype

```
void TblSetColumnMasked (TableType *tableP,
    Int16 column, Boolean masked)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *column*
Column number (zero-based).

→ *masked*
`true` to have the column be masked, `false` otherwise.

Returns Returns nothing.

Comments Masked cells are displayed as shaded. You should set a column to masked if its contents should be hidden when it contains information from a private database record and the user has set the display preference for private records to masked.

A table cell is not masked unless both its row and column are masked. This allows non-private information in the row item to remain visible. For example, the Datebook application shows the time for a private appointment, but it does not show the description.

Because the number of columns is static, you only need to call this function once per column when you first set up the table. The masked attribute on the row will determine if the contents of the table cell are actually displayed as shaded.

Compatibility Implemented only if 3.5 New Feature Set if present.

See Also `TblRowMasked()`, `TblSetRowMasked()`, `TableColumnAttrType`, `SecSelectViewStatus()`

TblSetColumnSpacing Function

Purpose Set the spacing after the specified `column`.

Declared In `Table.h`

Prototype

```
void TblSetColumnSpacing (TableType *tableP,
    Int16 column, Coord spacing)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *column*
Column number (zero-based).

→ *spacing*
Spacing after the column in pixels.

Returns Returns nothing.

This function may display a fatal error message if the `column` parameter is invalid.

See Also `TblSetColumnUsable()`, `TableColumnAttrType`

TblSetColumnUsable Function

Purpose Set a column in a table to usable or unusable.

Declared In `Table.h`

Prototype

```
void TblSetColumnUsable (TableType *tableP,
    Int16 column, Boolean usable)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *column*
Column number (zero-based).

→ *usable*
`true` for usable or `false` for not usable.

Returns Returns nothing.

This function may display a fatal error message if the `column` parameter is invalid.

Comments Columns that are not usable do not display.

See Also `TblMarkRowInvalid()`, `TableColumnAttrType`

TblSetColumnWidth Function

Purpose Set the width of the specified column.

Declared In `Table.h`

Prototype

```
void TblSetColumnWidth (TableType *tableP,
    Int16 column, Coord width)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *column*
Column number (zero-based).

→ *width*
Width of the column (in pixels).

Returns Returns nothing.

This function may display a fatal error message if the `column` parameter is invalid.

See Also `TblGetColumnWidth()`, `TableColumnAttrType`

TblSetCustomDrawProcedure Function

Purpose Set the custom draw callback procedure for the specified column.

Declared In `Table.h`

Prototype

```
void TblSetCustomDrawProcedure
    (TableType *tableP,
    Int16 column,
    TableDrawItemFuncPtr drawCallback)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *column*
Column number.

→ *drawCallback*
Callback function.

Returns Returns nothing.

Comments The custom draw callback function is used to draw table items with a `TableItemStyleType` of `customTableItem` or `tallCustomTableItem`. See the `TableItemType` description for more information.

This function may display a fatal error message if the `column` parameter is invalid.

See Also `TableDrawItemFuncType()`, `TblDrawTable()`, `TableColumnAttrType`

TblSetItemFont Function

Purpose Set the font used to display a table item.

Declared In `Table.h`

Prototype

```
void TblSetItemFont (TableType *tableP,
    Int16 row,
    Int16 column, FontID fontID)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number of the item (zero-based).

→ *column*
Column number of the item (zero-based).

→ *fontID*
ID of the font to be used.

Returns Returns nothing.

Comments This function sets the value stored in the `fontID` field for this table item. Only certain types of table items use the font specified by the `fontID` field when they are displayed. The `TableItemType` description specifies what font is used to display each type of table

item. It is not an error to set the `fontID` for a table item that does not use it.

This function may display a fatal error message if the `row` or `column` parameter specifies a row or column that is not on the screen.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `TblGetItemFont()`

TblSetItemInt Function

Purpose Set the integer value of the specified item.

Declared In `Table.h`

Prototype

```
void TblSetItemInt (TableType *tableP, Int16 row,
    Int16 column, Int16 value)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number of the item (zero-based).

→ *column*
Column number of the item (zero-based).

→ *value*
Any byte value (an integer).

Returns Returns nothing.

This function may display a fatal error message if the `row` or `column` parameter is invalid.

Comments You typically use this function when setting up and initializing a table for the first time to set the value of each table cell.

This function sets the value stored in the `intValue` field for this table item. Certain types of table items display the value stored in `intValue`, and other types display the value pointed to by the `ptr` field. See the `TableItemType` description for details. If you set the `intValue` of an item that displays its `ptr` value, it is not an error. An application can store whatever value it wants in the `intValue`

field; however, be aware that this has nothing to do with the value displayed by such a table cell.

See Also TblGetItemInt(), TblSetItemPtr()

TblSetItemPtr Function

Purpose Set the item to the specified pointer value.

Declared In `Table.h`

Prototype

```
void TblSetItemPtr (TableType *tableP, Int16 row,
    Int16 column, void *value)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number of the item (zero-based).

→ *column*
Column number of the item (zero-based).

→ *value*
Pointer to data to display in the table item.

Returns Returns nothing.

This function may display a fatal error message if the `row` or `column` parameter is invalid.

Comments This function sets the value stored in the `ptr` field for this table item. Certain types of table items display the value pointed to by `ptr`, and other types display the value stored in the `intValue` field. See the `TableItemType` description for details. If you set the `ptr` of an item that displays its `intValue`, it is not an error. An application can store whatever value it wants in the `ptr` field; however, be aware that this has nothing to do with the value displayed by such a table cell.

See Also `TblGetItemPtr()`, `TblSetItemInt()`

TblSetItemStyle Function

Purpose Set the type of item to display; for example, text, numbers, dates, and so on.

Declared In `Table.h`

Prototype

```
void TblSetItemStyle (TableType *tableP,
    Int16 row, Int16 column,
    TableItemStyleType type)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number of the item (zero-based).

→ *column*
Column number of the item (zero-based).

→ *type*
The type of item, such as an editable text field or a check box. See `TableItemType` for a list of possible values.

Returns Returns nothing.

This function may display a fatal error message if the `row` or `column` parameter is invalid.

Comments You typically use this function when first setting up and initializing a table; you do not dynamically change item styles.

Follow this function with a call to either `TblSetItemInt()` or `TblSetItemPtr()` to set the value displayed by the table item. You should call one or the other of these functions depending on the type of table item you specified. See the table in the `TableItemType` description for details.

Note also that a table column always contains items of the same type. For example, you might initialize a table using this code:

```
for (row = 0; row < rowsInTable; row++) {
  TblSetItemStyle (table, row, completedColumn,
    checkboxTableItem);
  TblSetItemStyle (table, row, priorityColumn,
    numericTableItem);
  TblSetItemStyle (table, row, descColumn,
    textTableItem);
  TblSetItemStyle (table, row, dueDateColumn,
```

```
    customTableItem);
  TblSetItemStyle (table, row, categoryColumn,
    customTableItem);
}
```

See Also TblSetCustomDrawProcedure()

TblSetLoadDataProcedure Function

Purpose Set the load-data callback procedure for the specified column.

Declared In `Table.h`

Prototype

```
void TblSetLoadDataProcedure (TableType *tableP,
    Int16 column,
    TableLoadDataFuncPtr loadDataCallback)
```

Parameters

→ *tableP*
Pointer to a table object. (See TableType.)

→ *column*
Column number (zero-based).

→ *loadDataCallback*
Callback procedure. See TableLoadDataFuncType().

Returns Returns nothing.

Comments The callback procedure is used to load the data values of a table item. See the TableLoadDataFuncType() for more information on writing the callback function.

You typically use this function when first setting up and initializing a table.

See Also TblSetCustomDrawProcedure()

TblSetRowData Function

Purpose Set the data value of the specified row. The data value is a placeholder for application-specific values.

Declared In `Table.h`

Prototype

```
void TblSetRowData (TableType *tableP, Int16 row,
    UInt32 data)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number (zero-based).

→ *data*
Application-specific data value to store for this row. For example, the Datebook and ToDo applications use this field to store the unique ID of the database record displayed by this row.

Returns Returns nothing.

This function may display a fatal error message if the `row` parameter is invalid.

See Also `TblGetRowData()`, `TblFindRowData()`

TblSetRowHeight Function

Purpose Set the height of the specified row.

Declared In `Table.h`

Prototype

```
void TblSetRowHeight (TableType *tableP,
    Int16 row, Coord height)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number (zero-based).

→ *height*
New height in pixels.

Returns Returns nothing.

This function may display a fatal error message if the `row` parameter is invalid.

See Also TblGetRowHeight(), TblSetRowStaticHeight()

TblSetRowID Function

Purpose Set the ID value of the specified row.

Declared In `Table.h`

Prototype

```
void TblSetRowID (TableType *tableP, Int16 row,
    UInt16 id)
```

Parameters

→ *tableP*
Pointer to a table object. (See TableType.)

→ *row*
Row number (zero-based).

→ *id*
ID to identify a row.

Returns Returns nothing.

This function may display a fatal error message if the `row` parameter is invalid.

See Also TblGetRowID(), TblFindRowID(), TableRowAttrType

TblSetRowMasked Function

Purpose Set a row in a table to masked or unmasked.

Declared In `Table.h`

Prototype

```
void TblSetRowMasked (TableType *tableP,
    Int16 row, Boolean masked)
```

Parameters

→ *tableP*
Pointer to a table object. (See TableType.)

→ *row*
Row number (zero-based).

→ *masked*
`true` to have the row be masked, `false` otherwise.

Returns Returns nothing.

Comments Masked cells are displayed as shaded. You should call this function before drawing or redrawing the table. If a table row contains a private database record and the user has set the display preference for private records to masked, you must call this function on that row. For example:

```
UInt16 attr;
privateRecordViewEnum privateRecordStatus;
Boolean masked;

privateRecordStatus = (privateRecordViewEnum)
  PrefGetPreference(prefShowPrivateRecords);
....
DmRecordInfo (ToDoDB, recordNum, &attr, NULL,
  NULL);
masked = ((attr & dmRecAttrSecret) &&
  (privateRecordStatus == maskPrivateRecords));
TblSetRowMasked(tableP, row, masked);
```

Note that a table cell is not masked unless both its row and column are masked. This allows non-private information in the row item to remain visible. For example, the Datebook application shows the time for a private appointment, but it does not show the description.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `TblRowMasked()`, `TblSetColumnMasked()`, `TableRowAttrType`, `SecSelectViewStatus()`

TblSetRowSelectable Function

Purpose Set a row in a table to selectable or nonselectable.

Declared In `Table.h`

Prototype

```
void TblSetRowSelectable (TableType *tableP,
    Int16 row, Boolean selectable)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number (zero-based).

→ *selectable*
`true` or `false`.

Returns Returns nothing.

This function may display a fatal error message if the `row` parameter is invalid.

Comments Rows that are not selectable don't highlight when touched.

See Also `TblRowSelectable()`, `TblSetRowUsable()`, `TableRowAttrType`

TblSetRowStaticHeight Function

Purpose Set the static height attribute of a row.

Declared In `Table.h`

Prototype

```
void TblSetRowStaticHeight (TableType *tableP,
    Int16 row, Boolean staticHeight)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number (zero-based).

→ *staticHeight*
`true` to set the static height, `false` to unset it.

Returns Nothing.

This function may display a fatal error message if the `row` parameter is invalid.

Comments A row that has its static height attribute set will not expand or contract the height of the row as text is added or removed from a text item.

Compatibility Implemented only if 2.0 New Feature Set is present.

TblSetRowUsable Function

Purpose Set a row in a table to usable or unusable. Rows that are not usable do not display.

Declared In `Table.h`

Prototype

```
void TblSetRowUsable (TableType *tableP,
    Int16 row, Boolean usable)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number (zero-based).

→ *usable*
`true` or `false`.

Returns Returns nothing.

This function may display a fatal error message if the `row` parameter is invalid.

See Also `TblRowUsable()`, `TblSetRowSelectable()`

TblSetSaveDataProcedure Function

Purpose Set the save-data callback procedure for the specified column.

Declared In `Table.h`

Prototype

```
void TblSetSaveDataProcedure (TableType *tableP,
    Int16 column,
    TableSaveDataFuncPtr saveDataCallback)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *column*
Column number (zero-based).

→ *saveDataCallback*
Callback function. See `TableSaveDataFuncType()`.

Returns Returns nothing.

This function may display a fatal error message if the `column` parameter is invalid.

Comments The callback procedure is called when the table object determines the data of a text object needs to be saved.

See Also `TblSetCustomDrawProcedure()`

TblSetSelection Function

Purpose Set a table item.

Declared In `Table.h`

Prototype

```
void TblSetSelection (TableType *tableP,
    Int16 row, Int16 column)
```

Parameters

→ *tableP*
Pointer to a `TableType`.

→ *row*
Table row.

→ *column*
Table column.

Returns `Returns nothing.`

Comments This function sets a table item, determined by the row and column arguments, as the current selection. `TblDrawTable()` must be called afterwards to update the UI.

Compatibility Implemented only if 4.0 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TblGlueSetSelection`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `TblGetNumberOfColumns()`, `TblGetTopRow()`

TblUnhighlightSelection Function

Purpose Unhighlight the currently selected item in a table.

Declared In `Table.h`

Prototype

```
void TblUnhighlightSelection (TableType *tableP)
```

Parameters → *tableP*
Pointer to a table object. (See `TableType`.)

Returns Returns nothing.

Application-Defined Functions

TableDrawItemFuncType Function

Purpose Draw a custom table item.

Declared In `Table.h`

Prototype

```
void TableDrawItemFuncType (void *tableP,
    Int16 row, Int16 column,
    RectangleType *bounds)
```

Parameters → *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number of the item to be drawn (zero-based).

→ *column*
Column number of the item to be drawn (zero-based).

→ *bounds*
The area of the screen in which the item is to be drawn.

Returns Returns nothing.

Comments This function is called during `TblDrawTable()` and `TblRedrawTable()`.

You implement a custom drawing function when your table contains items of type `customTableItem` or `tallCustomTableItem` (to draw the entire item) or `narrowTextTableItem` (to draw whatever is required in the space between the text and the right edge of the table cell).

You may implement a custom drawing function to include any style of information in the table. If you don't like the way a predefined item is drawn, you may prefer to use a `customTableItem` or `tallCustomTableItem` instead. For example, if you want to include a date in your table but you want it to show the year as well as the month and day, you should implement a custom drawing function.

See Also `TblSetCustomDrawProcedure()`, `TableItemType`

TableLoadDataFuncType Function

Purpose Load data into a column.

Declared In `Table.h`

Prototype

```
Err TableLoadDataFuncType (void *tableP,
    Int16 row, Int16 column, Boolean editable,
    MemHandle *dataH, Int16 *dataOffset,
    Int16 *dataSize, FieldPtr fld)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number of the table item to load.

→ *column*
Column number of the table item to load.

→ *editable*
If `true`, the table is currently being edited. If `false`, the table is being drawn but not necessarily being edited.

← *dataH*
Unlocked handle of a block containing a null-terminated text string.

← *dataOffset*
Offset from start of block to start of the text string.

← *dataSize*
Allocated size of text string, *not* the string length.

→ *fld*
Pointer to the text field in this table cell.

Returns Returns 0 upon success or an error if unsuccessful.

Comments This function is called in two cases: when a text field item is being drawn (`TblDrawTable()` or `TblRedrawTable()`) and when a text field item is being selected (part of `TblHandleEvent()`'s handling of `tblEnterEvent`). If this function returns an error (any nonzero value) and the item is being selected, then the item is not selected and the table's `editing` attribute is set to `false`.

You return the same values for `dataH`, `dataOffset`, and `dataSize` that you would pass to `FldSetText()`. That is, you can use this function to point the table cell's text field to a string in a database record so that you can edit that string directly using text field routines. To do so, return the handle to a database record in `dataH`, the offset from the start of the record to the start of the string in `dataOffset`, and the allocated size of the string in `dataSize`.

The handle that you return from this function is assumed to contain a null-terminated string starting at `dataOffset` bytes in the memory chunk. The string should be between 0 and `dataSize` - 1 bytes in length.

As with `FldSetText`, you are responsible for freeing the memory associated with the `dataH` parameter. You can do so in the `TableSaveDataFuncType()` function, but it is only called for a cell that has been edited. For non-editable text cells or text cells that are editable but were never selected, free the memory when you close the form.

The `fld` pointer passed to your function has already been initialized with default values by the table code. If you want to override a field's attributes (for example, if you want to change the underline mode) you can do so in this function.

See Also `TblDrawTable()`, `TblHandleEvent()`, `TableLoadDataFuncType()`

TableSaveDataFuncType Function

Purpose Save the data associated with a text field.

Declared In `Table.h`

Prototype

```
Boolean TableSaveDataFuncType (void *tableP,
    Int16 row, Int16 column)
```

Parameters

→ *tableP*
Pointer to a table object. (See `TableType`.)

→ *row*
Row number of the table item to load.

→ *column*
Column number of the table item to load.

Returns Return `true` if the table should be redrawn, or `false` if the table does not need to be redrawn.

Comments This is called before the memory associated with the currently selected text field in a table cell is freed. Implement this function if you need to do any special cleanup before this memory is freed.

This function is called only when the currently selected editable text field is releasing the focus. You can use `TblGetCurrentField()` to retrieve a pointer to this field. It is called only on the currently selected field, not on any other fields in the table.

Note that the table manager already disassociates the memory handle from the text field for you so that the memory associated with your data is not freed when the field is freed. The table manager also calls `FldCompactText()` for you.

If the text handle you returned in your `TableLoadDataFuncType()` callback points to a string on the dynamic heap, you should implement this callback function to store or free the handle. You can use `FldGetTextHandle()` to obtain the handle.

If you return `true` from this function, `TblRedrawTable()` is called. You should mark invalid any table rows that you want redrawn before returning.

See Also TblSetSaveDataProcedure()

21

UI Color List

This chapter provides information about the UI color list by discussing the following topics:

- UI Color Data Types
- UI Color Functions

The header file `UIColor.h` declares the API that this chapter describes. For more information on the color list, see "Color and Grayscale Support" on page 151 in the *Palm OS Programmer's Companion*, vol. I.

UI Color Data Types

UIColorTableEntries Enum

Purpose

The `UIColorTableEntries` enum declares symbolic color constants for the various UI elements.

Do not confuse the UI color list with the system color table. The **system color table** (or **system palette**) defines all available colors for the display or draw window, whether they are in use or not. The **UI color list** defines the colors used to draw the interface objects.

Prototype

```
typedef enum UIColorTableEntries {
  UIObjectFrame = 0,
  UIObjectFill,
  UIObjectForeground,
  UIObjectSelectedFill,
  UIObjectSelectedForeground,

  UIMenuFrame,
  UIMenuFill,
  UIMenuForeground,
  UIMenuSelectedFill,
  UIMenuSelectedForeground,
```

```
  UIFieldBackground,
  UIFieldText,
  UIFieldTextLines,
  UIFieldCaret,
  UIFieldTextHighlightBackground,
  UIFieldTextHighlightForeground,
  UIFieldFepRawText,
  UIFieldFepRawBackground,
  UIFieldFepConvertedText,
  UIFieldFepConvertedBackground,
  UIFieldFepUnderline,

  UIFormFrame,
  UIFormFill,

  UIDialogFrame,
  UIDialogFill,

  UIAlertFrame,
  UIAlertFill,

  UIOK,
  UICaution,
  UIWarning,

  UIFieldFepConvertedUnderline

  UILastColorTableEntry
} UIColorTableEntries;
```

Constants

`UIObjectFrame`
: Color for the border of user interface objects (such as command buttons, push buttons, selector triggers, menus, arrows checkboxes, and other controls).

`UIObjectFill`
: The background color for a solid or "filled" user interface object.

  Note that UI objects in tables use the `UIField`... colors instead of the `UIObject`... colors.

`UIObjectForeground`
: The color for foreground items (such as labels or graphics) in a user interface object.

`UIObjectSelectedFill`
: The background color of the currently selected user interface object, whether that object is solid or not.

`UIObjectSelectedForeground`
: The color of foreground items in a selected user interface object.

`UIMenuFrame`
: The color of the border around the menu.

`UIMenuFill`
: The background color of a menu item.

`UIMenuForeground`
: The color of the menu's text.

`UIMenuSelectedFill`
: The background color of a selected menu item.

`UIMenuSelectedForeground`
: The color of the text of a selected menu item.

`UIFieldBackground`
: The background color of an editable text field.

`UIFieldText`
: The color of the text in the editable field.

`UIFieldTextLines`
: The color of underlines in an editable field.

`UIFieldCaret`
: The color of the cursor in an editable text field.

`UIFieldTextHighlightBackground`
: The background color for selected text in an editable text field.

`UIFieldTextHighlightForeground`
: The color of the selected text in an editable text field.

`UIFieldFepRawText`
: Color used for unconverted text in the inline conversion area when a FEP is used as a text input method (for example, on Japanese devices).

  If the FEP colors are identical to field colors, unconverted text has a solid underline.

`UIFieldFepRawBackground`
: The background color for unconverted text in the inline conversion area when a FEP is used as a text input method.

  If the FEP colors are identical to field colors, unconverted text has a solid underline.

`UIFieldFepConvertedText`
: Color used for converted text in the inline conversion area when a FEP is used as a text input method (for example, on Japanese devices).

  If the FEP colors are identical to field colors, converted text has a double-thick underline.

`UIFieldFepConvertedBackground`
: The background color used for converted text in the inline conversion area.

  If the FEP colors are identical to field colors, converted text has a double-thick underline.

`UIFieldFepUnderline`
: The color used for underlines beneath unconverted text in the inline conversion area.

`UIFormFrame`
: The color of the border and titlebar on a form.

`UIFormFill`
: The background color of a form. White is recommended for this value.

`UIDialogFrame`
: The color of a border and titlebar on a modal form.

`UIDialogFill`
: The background color of a modal form.

`UIAlertFrame`
: The color of the border and titlebar on an alert panel.

`UIAlertFill`
: The background color of an alert panel.

`UIOK`
: Not used.

`UICaution`
: Not used.

`UIWarning`
: Not used.

`UIFieldFepConvertedUnderline`
: The color used for underlines beneath converted text in the inline conversion area.

`UILastColorTableEntry`
: Placeholder to indicate end of enum.

Compatibility Implemented only if 3.5 New Feature Set is present.

UI Color Functions

UIColorGetTableEntryIndex Function

Purpose Return the index value for a UI color for the current system palette.

Declared In `UIColor.h`

Prototype

```
IndexedColorType UIColorGetTableEntryIndex
    (UIColorTableEntries which)
```

Parameters

→ *`which`*
: One of the symbolic color constants. See `UIColorTableEntries`.

Returns Returns the system color table index of the color used for the specified symbolic color.

Comments One way to find out the indexes of all the colors that the OS is using is to loop through the UI color list, calling `UIColorGetTableEntryIndex` for each slot, and keep a list (excluding duplicates).

```
IndexedColorType
  colorsUsed[UILastColorTableEntry];
UInt16 numColors = 0;
...
for (i = 0; i < UILastColorTableEntry; i++) {
  IndexedColorType currentColor;
  Boolean isNew = true;

  currentColor = UIColorGetTableEntryIndex(i);

  for (j = 0; ((j < numColors) && isNew); j++)
    if (colorsUsed[j] == currentColor)
      isNew = false; /* exit loop */
  if (isNew) {
    numColors++;
    colorsUsed[j] = currentColor;
  }
```

To get the RGB values of the colors, do the same thing but call `UIColorGetTableEntryRGB()`.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `IndexedColorType`, `WinIndexToRGB()`

UIColorGetTableEntryRGB Function

Purpose Return the RGB value for the UI color.

Declared In `UIColor.h`

Prototype

```
void UIColorGetTableEntryRGB
    (UIColorTableEntries which,
    RGBColorType *rgbP)
```

Parameters

→ *which*
One of the symbolic color constants. See `UIColorTableEntries`.

← *rgbP*
Pointer to an RGB color value corresponding to the current color used for the symbolic color. (See `RGBColorType`.)

Returns Returns nothing.

Comments In general, it is more efficient to work with indexed color entries instead of RGB color entries.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `UIColorGetTableEntryIndex()`, `WinRGBToIndex()`

UIColorSetTableEntry Function

Purpose Change a value in the UI color list.

Declared In `UIColor.h`

Prototype

```
Err UIColorSetTableEntry
    (UIColorTableEntries which,
    const RGBColorType *rgbP)
```

Parameters

→ *which*
One of the symbolic color constants. See `UIColorTableEntries`.

→ *rgbP*
The RGB value of the color that should be used for the specified UI object. (See `RGBColorType`.)

Returns Returns 0 upon success.

Comments Sets the value of a UI color entry to the passed RGB value. Updates the index for that UI color entry to the current best fit for that RGB value according to the palette used by the current draw window.

It is best to use this function only if the draw window is currently onscreen. Otherwise, the best-fit algorithm may choose a color that is not available on the current screen.

See Also `WinIndexToRGB()`, `UIColorGetTableEntryIndex()`, `UIColorGetTableEntryRGB()`

22

UI Controls

This chapter describes the UI controls API as declared in `UIControls.h`.

UI Control Functions

UIBrightnessAdjust Function

Purpose Displays the brightness adjust dialog.

Declared In `UIControls.h`

Prototype `void UIBrightnessAdjust()`

Parameters `None`

Returns Returns nothing.

Comments On hardware that supports a brightness setting, this function displays a dialog that allows the user to change the brightness level. On hardware that has a backlight, this function toggles the backlight.

Compatibility Implemented only if 3.5 New Feature Set is present.

UIContrastAdjust Function

Purpose Displays the contrast adjust dialog (currently only available on the Palm V™ Connected Organizer).

Declared In `UIControls.h`

Prototype `void UIContrastAdjust()`

Parameters `None.`

Returns Returns nothing.

Compatibility This function was renamed from `ContrastAdjust` to `UIContrastAdjust` in Palm OS® release 3.5. The `ContrastAdjust` function is available if 3.1 New Feature Set is present.

UIPickColor Function

Purpose Displays a dialog that allows the user to choose a color.

Declared In `UIControls.h`

Prototype

```
Boolean UIPickColor (IndexedColorType *indexP,
    RGBColorType *rgbP,
    UIPickColorStartType start,
    const Char *titleP, const Char *tipP)
```

Parameters

↔ *indexP*
: Index value of the selected color. (See `IndexedColorType`.) Upon entry, this points to the index value of the color initially selected. Upon return, this points to the index value of the color the user selected. Pass `NULL` to not set or return this value.

↔ *rgbP*
: RGB value of the selected color. (See `RGBColorType`.) Upon entry, this points to the RGB value of the color initially selected when the dialog is displayed. Upon return, this points to the RGB value that the user selected. Pass `NULL` to not set or return this value.

→ *start*
: Either `UIPickColorStartPalette` to display the system palette as a series of color squares or `UIPickColorStartRGB` to display individual sliders for the red, green, and blue values. This parameter is only used if both `indexP` and `rgbP` are not `NULL`.

→ *titleP*
: String to display as the title of the dialog. Specify `NULL` to use the default title, which is "Pick Color."

→ *tipP*
: Not used.

Returns Returns `true` if a new color was selected, `false` otherwise.

Comments Use this function to allow users to choose a color used in your user interface. (The system never calls `UIPickColor`.)

This function can display two versions of the dialog: palette or RGB. The palette version of the dialog displays a series of squares, each containing a different color defined on the system palette. The `indexP` value contains the index of the square that is initially selected.

The RGB version of the dialog displays three sliders that allow the user to select the level of red, green, and blue in the color. The `rgbP` parameter contains the red, green, and blue values initially shown in the dialog. The sliders only allow values that are defined in the current system color table.

If `indexP` is initially `NULL`, only the RGB dialog is displayed. Similarly, if `rgbP` is `NULL`, only the palette version is displayed. If both parameters are non-`NULL`, the system adds a pull-down list that allows the user to switch between the palette dialog and the RGB dialog, and the `start` parameter controls which version of the dialog is initially shown. In this case, both `indexP` and `rgbP` contain the value of the user-selected color upon return.

Palm OS 3.5 supports a maximum of 256 colors. The number of possible RGB colors greatly exceeds this amount. For this reason, the chosen RGB may not have an exact match. If this is the case, the `indexP` parameter (if not `NULL`) contains the closest match using a luminance best-fit if the color lookup table is entirely grayscale (red, green, and blue values for each entry are identical), or a shortest-distance fit in the RGB space is used if the palette contains colors.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinSetBackColor()`, `WinSetForeColor()`, `WinSetTextColor()`, `UIColorSetTableEntry()`

23

Miscellaneous User Interface Functions

This chapter provides descriptions of miscellaneous user interface functions. It covers the following topics:

- Miscellaneous User Interface Data Structures
- Miscellaneous User Interface Functions

You can find declarations for the functions described in this chapter in the header files `AppLaunchCmd.h`,`PhoneLookup.h`, and `UIResources.h`.

Miscellaneous User Interface Data Structures

The `PhoneNumberLookupCustom()` function uses these data structures to look up contact information based upon the current cursor position.

AddressLookupFields Enum

Purpose

The `AddressLookupFields` enum specifies the fields you can search by and the corresponding fields to return using the `field1` and `field2` elements of the `AddrLookupParamsType` structure. For both `field1` and `field2` pass one of the values up to, but not including, `addrLookupFieldCount`.

Prototype

```
typedef enum {
  addrLookupName,
  addrLookupFirstName,
  addrLookupCompany,
  addrLookupAddress,
  addrLookupCity,
  addrLookupState,
  addrLookupZipCode,
```

```
  addrLookupCountry,
  addrLookupTitle,
  addrLookupCustom1,
  addrLookupCustom2,
  addrLookupCustom3,
  addrLookupCustom4,
  addrLookupNote,
  addrLookupWork,
  addrLookupHome,
  addrLookupFax,
  addrLookupOther,
  addrLookupEmail,
  addrLookupMain,
  addrLookupPager,
  addrLookupMobile,
  addrLookupSortField,
  addrLookupListPhone,
  addrLookupFieldCount,

  addrLookupNoField = 0xff
} AddressLookupFields;
```

AddrLookupParamsType Struct

Purpose Pass this structure to PhoneNumberLookupCustom() to precisely control the phone number lookup dialog and paste process.

Prototype

```
typedef struct {
  Char *title;
  Char *pasteButtonText;
  Char lookupString[addrLookupStringLength];
  AddressLookupFields field1;
  AddressLookupFields field2;
  Boolean field2Optional;
  Boolean userShouldInteract;
  Char *formatStringP;
  MemHandle resultStringH;
  UInt32 uniqueID;
} AddrLookupParamsType;
typedef AddrLookupParamsType
*AddrLookupParamsPtr;
```

Fields

`title`
Title to appear in the title bar. Supply `NULL` to use the default title.

`pasteButtonText`
Text to appear in paste button. Supply `NULL` to use the default, "paste".

`lookupString`
Buffer containing the string to look up. If the string matches only one record, that record is used without presenting the lookup dialog to the user. `PhoneNumberLookup()` and `PhoneNumberLookupCustom()` both set this field based upon the current selection or cursor position.

`field1`
Field to search by. This field appears on the left side of the lookup dialog. If the field is the sort field, the search is performed using a binary search. If the field isn't the sort field, searching is performed by a linear search, which can be slow. Supply one of the values in the `AddressLookupFields` enum to specify the field to search by.

`field2`
Field to display on the right. Often displays some information about the person. If it is a phone field and a record has multiple instances of the phone type then the person appears once per instance of the phone type. Either `field1` or `field2` may be a phone field, but not both. Supply one of the values in the `AddressLookupFields` enum to specify the field to display.

`field2Optional`
A value of `true` means that the record need not have `field2` in order to be listed. A value of `false` indicates that `field2` is required in order for the record to be listed.

`userShouldInteract`
A value of `true` forces the user to resolve non-unique lookups. A `false` value means a non-unique and complete lookup causes `resultStringH` to be set to `NULL` and `uniqueID` to be set to 0.

`formatStringP`

Controls the format of the paste string. All characters in the format string are literal unless they identify a field (signified by a caret (^) followed by the field name). For example, the format string "^first - ^home" might result in "Roger - 123-4567". Allowable field names are:

```
name
first
company
address
city
state
zipcode
country
title
custom1
custom2
custom3
custom4
work
home
fax
other
email
main
pager
mobile
listname
```

`resultStringH`

If there is a format string, a result string is allocated on the dynamic heap and its handle is returned here.

`uniqueID`
The unique ID of the found record, or 0 if none was found, is returned here.

Miscellaneous User Interface Functions

PhoneNumberLookup Function

Purpose Calls the Address Book application to look up a phone number.

Declared In `PhoneLookup.h`

Prototype
```
void PhoneNumberLookup (FieldType *fldP)
```

Parameters → *`fldP`*
Field object in which the text to match is found.

Returns Nothing returned; it's locked.

Comments This function displays the user's phone list and inserts the chosen name and number (or company name, name, and number, if that's how the user's Address Book preferences indicate that the phone list should be sorted) into the specified field. When displaying the phone list, `PhoneNumberLookup` scrolls the list to that entry that best matches the supplied field. The match compares the field contents against the name or company name (depending on the user's preferences) as follows:

- If the field contains selected text, `PhoneNumberLookup` tries to match against the selected text. The selected text is then replaced with the text of the chosen address list entry.
- If there is no selected text in the field, `PhoneNumberLookup` matches against the word in which the cursor lies (the match will take place if the cursor is at the beginning, the end, or within a word). The matched word is replaced with the text of the chosen address list entry.
- If the cursor does not lie within or adjoin a word, `PhoneNumberLookup` displays the address list starting at the first entry, and the text of the chosen entry is inserted at the current position within the text field.

If the user chooses Cancel when the address list is displayed, the field contents are left unaltered. The paste operation takes place through the clipboard so that Undo can be used to restore the field to its previous state.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `PhoneNumberLookupCustom()`

PhoneNumberLookupCustom Function

Purpose Calls the Address Book application to look up a phone number.

Declared In `PhoneLookup.h`

Prototype

```
void PhoneNumberLookupCustom (FieldType *fldP,
    AddrLookupParamsType *params,
    Boolean useClipboard)
```

Parameters

→ *fldP*
Field object in which the text to match is found.

↔ *params*
A structure that allows full control over the find dialog and the format of the resulting paste string. See `AddrLookupParamsType` for a description of the fields in this structure.

→ *useClipboard*
If true, `PhoneNumberLookupCustom` pastes the result into the field through the clipboard, thereby enabling undo.

Returns Nothing returned; it's locked.

Comments This function displays two fields from each record in the user's address list and inserts a formatted string based upon fields in the chosen record into the specified field. When displaying the address list, `PhoneNumberLookupCustom` scrolls the list to that entry that best matches the supplied field. The match compares the field contents against the field specified in the `params` structure's `field1` element as follows:

- If the field contains selected text, `PhoneNumberLookup` tries to match against the selected text. The selected text is then replaced with the text of the chosen address list entry.

- If there is no selected text in the field, `PhoneNumberLookup` matches against the word in which the cursor lies (the match will take place if the cursor is at the beginning, the end, or within a word). The matched word is replaced with the text of the chosen address list entry.
- If the cursor does not lie within or adjoin a word, `PhoneNumberLookup` displays the address list starting at the first entry, and the text of the chosen entry is inserted at the current position within the text field.

PhoneNumberLookupCustom copies the portion of the field used for the search—the selected text or the word in which the cursor lies—into the `lookupString` field in the `params` structure prior to replacing that part of the field with the user-selected entry.

If the user chooses Cancel when the address list is displayed, the field contents are left unaltered. Depending on the value of the `useClipboard` parameter, the paste operation can take place through the clipboard so that Undo can be used to restore the field to its previous state.

Compatibility Implemented only if 4.0 New Feature Set is present.

ResLoadConstant Function

Purpose Load a constant from a `'tint'` resource and return its value.

Declared In `UIResources.h`

Prototype

```
UInt32 ResLoadConstant (UInt16 rscID)
```

Parameters

→ *`rscID`*
The ID of the `'tint'` resource (symbolically named `constantRscType`) to load.

Returns The four-byte value of the constant in the resource, or 0 if the resource could not be found. The return value may be cast as necessary.

Comments Use this function to load "soft constants"—constant values that are stored as `'tint'` resources. (All open resource databases are searched for the resource ID you specify.) You should store a constant value as a resource when its value changes depending on the locale.

As an example, consider the maximum length of the Alarm Sound trigger label in the Datebook application's preferences panel. The list displayed by this trigger uses the localized name for each sound stored in the system. Because localized names are used, the maximum length that the Datebook application allows for the label differs depending on the current locale. The maximum length is stored as a resource constant so that each overlay database can specify a different value for the constant.

Compatibility Implemented only if 3.5 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS®, link with the PalmOSGlue library and call `ResGlueLoadConstant`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `DmGetResource()`, `DmGet1Resource()`

ResLoadForm Function

Purpose Copy and initialize a form resource. The structures are complete except pointers updating. Pointers are stored as offsets from the beginning of the form.

Declared In `UIResources.h`

Prototype

```
void *ResLoadForm (UInt16 rscID)
```

Parameters

→ *rscID*
The resource ID of the form.

Returns The handle of the memory block that the form is in, since the form structure begins with the `WindowType`, this is also a `WinHandle`.

Compatibility If either 5.0 New Feature Set or Palm OS Cobalt Feature Set is present this function is unimplemented.

ResLoadMenu Function

Purpose Copy and initialize a menu resource. The structures are complete except pointers updating. Pointers are stored as offsets from the beginning of the menu.

Declared In `UIResources.h`

Prototype

```
void *ResLoadMenu (UInt16 rscID)
```

Parameters → *rscID*

The resource ID of the menu.

Returns The handle of the memory block that the form is in, since the form structure begins with the `WindowType` this is also a `WinHandle`.

Part II
System Management

24

Alarm Manager

This chapter provides reference material for the alarm manager:

- Alarm Manager Functions
- Application-Defined Functions

The alarm manager API is declared in the header file `AlarmMgr.h`.

For more information on the Alarm Manager, see the section "Alarms" in the *Palm OS Programmer's Companion*, vol. I.

Alarm Manager Functions

AlmGetAlarm Function

Purpose Return the date and time for the application's alarm, if one is set.

Declared In `AlarmMgr.h`

Prototype

```
UInt32 AlmGetAlarm (UInt16 cardNo, LocalID dbID,
   UInt32 *refP)
```

Parameters

→ *cardNo*
Number of the storage card on which the application resides.

→ *dbID*
Local ID of the application.

← *refP*
The alarm's reference value is returned here. This value is passed to the application when the alarm is triggered.

Returns The date and time the alarm will trigger, given in seconds since 1/1/1904; if no alarm is active for the application, 0 is returned for the alarm seconds and the reference value is undefined.

See Also `AlmSetAlarm()`

AlmGetProcAlarm Macro

Purpose Returns the date and time that a procedure alarm will trigger. Also returns the caller-defined alarm reference value.

Declared In `AlarmMgr.h`

Prototype

```
#define AlmGetProcAlarm (procP, refP)
```

Parameters

→ *procP*
Pointer to a function that will be called when alarm is triggered. See `AlmAlarmProcPtr()`.

← *refP*
A `UInt32` pointer to a location where the alarm's reference value is returned. This value is passed to the procedure when the alarm is triggered.

Returns The date and time the alarm will trigger, given in seconds since 1/1/1904; if no alarm is active for the procedure, 0 is returned for the alarm seconds and the reference value is undefined.

Compatibility Implemented only if 3.2 New Feature Set is present.

See Also `AlmSetProcAlarm()`

AlmSetAlarm Function

Purpose Set or cancel an alarm for the given application.

Declared In `AlarmMgr.h`

Prototype

```
Err AlmSetAlarm (UInt16 cardNo, LocalID dbID,
    UInt32 ref, UInt32 alarmSeconds,
    Boolean quiet)
```

Parameters

→ *cardNo*
Number of the storage card on which the application resides.

→ *dbID*
Local ID of the application.

→ *ref*
Caller-defined value. This value is passed with the launch code that notifies the application that the alarm has been triggered.

→ *alarmSeconds*
: Alarm date/time in seconds since 1/1/1904, or 0 to cancel the current alarm (if any).

→ *quiet*
: Reserved for future upgrade. This value is not currently used.

Returns

`0`
: No error.

`almErrMemory`
: Insufficient memory.

`almErrFull`
: Alarm table is full.

Comments

This function sets an alarm for the specified application. An application can have only one alarm set at a time. If an alarm for this application has already been set, it is replaced with the new alarm.

The `alarmSeconds` parameter specifies the time at which the alarm will be triggered. As soon as possible after this time, the alarm manager sends the `sysAppLaunchCmdAlarmTriggered` launch code to the specified application. If there is another alarm that should be set for this application, you can set it in response to this launch code. Following the `sysAppLaunchCmdAlarmTriggered` launch code, the alarm manager sends a `sysAppLaunchCmdDisplayAlarm` launch code. This is where your application should do things such as display a modal dialog indicating that the event has occurred. Read more about these launch codes in Chapter 1, "Application Launch Codes."

If your application needs access to any particular value to respond to the alarm, pass a pointer to that value in the `ref` parameter. The system will pass this pointer back to the application in the launch codes' parameter blocks.

See Also

`AlmGetAlarm()`

AlmSetProcAlarm Macro

Purpose Sets or cancels a procedure alarm.

Declared In `AlarmMgr.h`

Prototype

```
#define AlmSetProcAlarm (procP, ref, alarmSeconds)
```

Parameters

→ *procP*
: Pointer to a function that should be called when alarm is triggered. See `AlmAlarmProcPtr()`.

→ *ref*
: A caller-defined `UInt32` value. This value is passed with the launch code that notifies the application that the alarm has been triggered.

→ *alarmSeconds*
: A `UInt32` indicating the alarm date/time in seconds since 1/1/1904, or 0 to cancel the current alarm (if any).

Returns Returns one of the following error codes:

`0`
: No error.

`almErrMemory`
: Insufficient memory.

`almErrFull`
: Alarm table is full.

Comments This macro is similar to the `AlmSetAlarm()` function, but it specifies a procedure to be called at the specified date and time rather than an application to be launched. With this macro, you can set alarms that are independent of any application. For example, a shared library can set procedure alarms that call a procedure implemented in the library.

Procedure alarms also differ from regular system alarms in that if they trigger when the device is in sleep mode, the LCD does not light up. Thus, you can use procedure alarms to perform a scheduled task in a manner that is entirely hidden from the user.

IMPORTANT: Because the `procP` pointer is used to directly call the procedure, the pointer must remain valid from the time `AlmSetProcAlarm` is called to the time the alarm is triggered. If the procedure is in a shared library, you must keep the library open. If the procedure is in a separately loaded code resource, the resource must remain locked until the alarm is triggered. When you close a library or unlock a resource, you must remove any pending alarms. If you don't, the system will crash when the alarm is triggered.

Compatibility Implemented only if 3.2 New Feature Set is present.

See Also `AlmGetProcAlarm()`

Application-Defined Functions

AlmAlarmProcPtr Function

Purpose A procedure to be executed when an alarm is triggered.

Declared In `AlarmMgr.h`

Prototype

```
void (*AlmAlarmProcPtr) (UInt16 almProcCmd,
    SysAlarmTriggeredParamType *paramP)
```

Parameters

→ *almProcCmd*

One of the `AlmProcCmdEnum` constants. These are commands that your function must handle. Possible values are:

`almProcCmdTriggered`
: The alarm's date and time has passed and the alarm has been triggered. The function should perform its main task in response to this command.

`almProcCmdReschedule`
: A system time change occurred, so the function must reschedule the alarm.

→ *paramP*

Pointer to a `SysAlarmTriggeredParamType` structure. See below.

Returns Returns nothing.

Comments `AlmAlarmProcPtr` procedures are called when an alarm set by `AlmSetProcAlarm()` is triggered. Your implementation should check the value of `almProcCmd` and respond accordingly.

The `paramP` parameter is a pointer to a `SysAlarmTriggeredParamType` structure. This structure is defined as:

```
typedef struct SysAlarmTriggeredParamType {
  UInt32   ref;
  UInt32   alarmSeconds;
  Boolean purgeAlarm;
} SysAlarmTriggeredParamType;
```

`ref` and `alarmSeconds` are both values specified in `AlmSetProcAlarm()` when the alarm is set. The `purgeAlarm` field specifies if the alarm will be removed from the alarm table when the function returns so that the `sysAppLaunchCmdDisplayAlarm` launch code is not triggered. This should be `true` for all procedure alarms; the alarm manager set it to `true` for you after your function returns.

If necessary, you can define new values for the `almProcCmd` parameter to call the procedure for something other than a triggered alarm or a system time change. The value you use must be greater than the constant `almProcCmdCustom` as defined in `AlarmMgr.h`.

Compatibility Implemented only if 3.2 New Feature Set is present.

See Also `AlmGetProcAlarm()`

25

ARM Debug Nub

This chapter provides documentation for the ARM-based debugger nub. It is divided into the following sections:

ARM Debug Nub Structures and Types 516
ARM Debug Nub Constants 516
ARM Debug Nub Functions and Macros 519

The header file `AdnDebugMgr.h` declares the API that this chapter describes. These functions can be used within ARM subroutines, also called PACE native objects ("PNOs").

For more information on using the APIs described in this reference chapter, see "Using the ARM Debug Nub" on page 426 of Chapter 15, "Debugging Strategies."

ARM Debug Nub Structures and Types

AdnDbgPostLoadParamsType Struct

Purpose Parameter block that accompanies the `_SemihostPostLoad()` function. This structure is not normally used by third-party developers.

Declared In `AdnDebugMgr.h`

Prototype

```
typedef struct _AdnDbgPostLoadParamsType {
   int32_t processID;
   uint32_t moduleID;
   void *codeAddr;
   void *dataAddr;
   uint32_t libType;
   uint32_t libCreator;
   uint32_t rsrcType;
   uint16_t rsrcID;
   uint16_t reserved;
} AdnDbgPostLoadParamsType
```

AdnDbgPostUnloadParamsType Struct

Purpose Parameter block that accompanies the `_SemihostPostUnload()` function. This structure is not normally used by third-party developers.

Declared In `AdnDebugMgr.h`

Prototype

```
typedef struct _AdnDbgPostUnloadParamsType {
   uint32_t processID;
   uint32_t moduleID;
} AdnDbgPostUnloadParamsType
```

ARM Debug Nub Constants

Debugging Feature Flags

Purpose Flags supplied to `AdnDebugEnableSet()` and returned by `AdnDebugEnableGet()` and `AdnDebugEnableGetSupported()` that specify which debugging features are enabled.

Declared In `AdnDebugMgr.h`

Constants

`#define kAdnEnableDebugIndicator 0x00000002`

The debug indicator enables a visual indicator that shows when the debugger nub is performing or waiting for serial communication. For example, on the Tungsten T device, the green LED blinks to indicate that the debugger nub is active.

`#define kAdnEnableFullDebugging 0x00000004`

The full debugging feature must be enabled in order for application calls to `AdnDebugNativeRegister()` to be processed correctly. If this feature is disabled, then other native debugging calls (such as ModulePostLoad and ModulePostUnload) are also disabled.

`#define kAdnEnableMasterSwitch 0x00000001`

When the master switch is enabled, the debug nub catches fatal ARM exceptions, enters the debugger if `AdnDebugBreak()` is called, and catches `ErrFatalDisplay()` calls.

When the master switch is disabled, the you are limited to calling the `AdnDebugEnableSet()` and `AdnDebugEnableGet()` macros.

`#define kAdnEnableShowSafeFatalAlerts 0x00000008`

When show safe fatal alerts is enabled, the debugger nub does not catch calls to `ErrFatalDisplay()` if the Reset, Debug, and Continue options are displayed.

Semihosting SWI Opcodes

Purpose Opcodes used to invoke various functions. Applications do not need to make use of these opcodes directly; the various macros described under "ARM Debug Nub Functions and Macros" on page 519 use them to access the needed operating system functions.

Declared In `AdnDebugMgr.h`

Constants `#define kAdnSemihostArmReportException 0x0018`

```
#define kAdnSemihostArmWrite0 0x0004
#define kAdnSemihostDebugEnableGet 0x0153
#define kAdnSemihostDebugEnableGetSupported 0x0155
#define kAdnSemihostDebugEnableSet 0x0152
#define kAdnSemihostLicenseeSpecific 0x0154
#define kAdnSemihostNativeRegister 0x0150
#define kAdnSemihostNativeUnregister 0x0151
#define kAdnSemihostPUDConnect 0x0101
#define kAdnSemihostPUDDebugBreak 0x0132
#define kAdnSemihostPUDDownloadEvent 0x0140
#define kAdnSemihostPUDFaultNotification 0x0133
#define kAdnSemihostPUDModulePostLoad 0x0130
#define kAdnSemihostPUDModulePostUnload 0x0131
#define kAdnSemihostPUDModuleTableUpdate 0x0134
#define kAdnSemihostPUDProcessCreate 0x0110
#define kAdnSemihostPUDProcessDestroy 0x0111
#define kAdnSemihostPUDProfilerEvent 0x0141
#define kAdnSemihostPUDThreadCreate 0x0120
#define kAdnSemihostPUDThreadDestroy 0x0121
```

Miscellaneous ARM Debug Nub Constants

Purpose Additional constants declared by the ARM debug nub header file `AdnDebugMgr.h`.

Declared In `AdnDebugMgr.h`

Constants `#define adnFtrCreator 'adbg'`

Creator ID of the ARM debug nub feature; if this feature is present, the value returned from a call to `FtrGet()` is the version of the ARM debug nub feature present on the device. See "ARM Debug Nub Feature Set" on page 2440 for an example of how to use this constant.

`#define adnFtrNumVersion 0`
: Feature number that retrieves the version of the ARM debug nub feature. See "ARM Debug Nub Feature Set" on page 2440 for an example of how to use this constant.

`#define kAdnErrUnsupportedCall 0xFFFFFFFF`
: Value returned by `AdnDebugLicenseeSpecific()` if the requested function is not supported by the device.

ARM Debug Nub Functions and Macros

AdnDebugBreak Macro

Purpose Break into Palm OS Debugger from the ARM subroutine.

Declared In `AdnDebugMgr.h`

Prototype
```
#define AdnDebugBreak ()
```

Parameters None.

Returns Nothing.

Comments This macro is the ARM-based equivalent to the Debug Manager call `DbgBreak()`.

Compatibility Implemented only on devices running Palm OS Garnet or Palm OS Cobalt where the ARM Debug Nub Feature Set is present.

AdnDebugEnableGet Macro

Purpose Determines which debugging features are enabled.

Declared In `AdnDebugMgr.h`

Prototype
```
#define AdnDebugEnableGet ()
```

Parameters None.

Returns A `UInt32` representing flags as defined under "Debugging Feature Flags" on page 516.

Example The following code obtains the current debugger settings and then updates them by setting addtional flags.

```
UInt32 flags = AdnDebugEnableGet();
flags |= kAdnEnableMasterSwitch | kAdnEnableFullDebugging;
AdnDebugEnableSet(flags);
```

See Also `AdnDebugEnableGet()`, `AdnDebugEnableSet`

Compatibility Implemented only on devices running Palm OS Garnet or Palm OS Cobalt where the ARM Debug Nub Feature Set is present.

AdnDebugEnableGetSupported Macro

Purpose Determines which debugging features are supported.

Declared In `AdnDebugMgr.h`

Prototype

```
#define AdnDebugEnableGetSupported ()
```

Parameters None.

Returns A `UInt32` representing flags as defined under "Debugging Feature Flags" on page 516.

See Also `AdnDebugEnableGet()`

Compatibility Implemented only on devices running Palm OS Garnet or Palm OS Cobalt where the ARM Debug Nub Feature Set is present.

AdnDebugEnableSet Macro

Purpose Enables the specified debugging features.

Declared In `AdnDebugMgr.h`

Prototype

```
#define AdnDebugEnableSet (flags)
```

Parameters

→ *flags*

`kAdnEnable...` flags as defined under "Debugging Feature Flags" on page 516

Returns Nothing.

Comments The ARM debugger nub is not active on an ARM-based device until an application activates it. The ARM subroutine must first enable the ARM debugger nub by calling this macro.

See Also `AdnDebugEnableGet`

Compatibility Implemented only on devices running Palm OS Garnet or Palm OS Cobalt where the ARM Debug Nub Feature Set is present.

AdnDebugLicenseeSpecific Macro

Purpose Makes a licensee-specific call to the ARM debug nub.

Declared In `AdnDebugMgr.h`

Prototype

```
#define AdnDebugLicenseeSpecific (oemID,
    selector, param)
```

Parameters

→ *oemID*
PalmSource-registered OEM ID (creator ID).

→ *selector*
Licensee-specific function selector.

→ *param*
Function-specific parameter.

Returns Returns `kAdnErrUnsupportedCall` if the licensee-specific function in not supported.

Compatibility Implemented only on devices running Palm OS Garnet or Palm OS Cobalt where the ARM Debug Nub Feature Set is present.

AdnDebugMessage Macro

Purpose Displays a debug message in the debugger.

Declared In `AdnDebugMgr.h`

Prototype

```
#define AdnDebugMessage (messageP)
```

Parameters

→ *messageP*
A pointer to a sequence of ASCII characters terminated by a null character (the string that is displayed in the debugger).

Returns Nothing.

Comments This macro is the ARM-based equivalent to the Debug Manager call `DbgMessage()`.

Compatibility Implemented only on devices running Palm OS Garnet or Palm OS Cobalt where the ARM Debug Nub Feature Set is present.

AdnDebugMessageIf Macro

Purpose Displays a debug message in the debugger if a specified condition is `true`.

Declared In `AdnDebugMgr.h`

Prototype

```
#define AdnDebugMessageIf (condition, messageP)
```

Parameters

→ *condition*
A boolean value.

→ *messageP*
A pointer to a sequence of ASCII characters terminated by a null character (the string that is displayed in the debugger).

Returns Nothing.

See Also `AdnDebugMessage`

Compatibility Implemented only on devices running Palm OS Garnet or Palm OS Cobalt where the ARM Debug Nub Feature Set is present.

AdnDebugNativeRegister Macro

Purpose Registers a PACE native object (PNO) with the debugger.

Declared In `AdnDebugMgr.h`

Prototype

```
#define AdnDebugNativeRegister (_dbType,
    _dbCreator, _rsrcType, _rsrcID)
```

Parameters

→ *_dbType*
The application database type (for example, `'appl'`).

→ *_dbCreator*
The application database's creator code.

→ *_rsrcType*
The PNO's resource type (for example, `'ARMC'`).

→ *_rsrcID*
The PNO's resource ID.

Returns Nothing.

Comments PNO registration should be done after locking the code. Any previously unresolved breakpoints will be activated.

You must have set the `kAdnEnableFullDebugging` flag using `AdnDebugEnableSet()` in order for calls to `AdnDebugNativeRegister()` to be processed correctly.

See Also `AdnDebugNativeRegisterAddr()`, `AdnDebugNativeUnregister()`

Compatibility Implemented only on devices running Palm OS Garnet or Palm OS Cobalt where the ARM Debug Nub Feature Set is present.

AdnDebugNativeRegisterAddr Macro

Purpose Registers a PACE native object (PNO) with the debugger. This version can be used for code that is in a chunk separate from the application. Most PNOs can use the simpler form, `AdnDebugNativeRegister()`.

Declared In `AdnDebugMgr.h`

Prototype

```
#define AdnDebugNativeRegisterAddr (_dbType,
    _dbCreator, _rsrcType, _rsrcID, _codeAddr)
```

Parameters

→ *_dbType*
The application database type (for example, `'appl'`).

→ *_dbCreator*
The application database's creator code.

→ *_rsrcType*
The PNO's resource type (for example, `'ARMC'`).

→ *_rsrcID*
The PNO's resource ID.

→ *_codeAddr*
The PNO's code base address.

Returns Nothing.

Comments You must have set the `kAdnEnableFullDebugging` flag using `AdnDebugEnableSet()` in order for calls to `AdnDebugNativeRegister()` to be processed correctly.

See Also `AdnDebugNativeRegister()`, `AdnDebugNativeUnregisterAddr()`

Compatibility Implemented only on devices running Palm OS Garnet or Palm OS Cobalt where the ARM Debug Nub Feature Set is present.

AdnDebugNativeUnregister Macro

Purpose Ask the debugger to unregister the PACE native object (PNO) invoking this macro.

Declared In `AdnDebugMgr.h`

Prototype

```
#define AdnDebugNativeUnregister ()
```

Parameters None.

Returns Nothing.

Comments You should call this macro prior to unlocking the PNO code resource so that the debugger breakpoints can be removed.

To unregister a PNO other than the one invoking this macro, use `AdnDebugNativeUnregisterAddr()`. `AdnDebugNativeUnregister()` invokes `AdnGetNativeCodeBaseAddr()` in order to determine the base address of the PNO to be unregistered.

See Also `AdnDebugNativeRegister()`, `AdnDebugNativeUnregisterAddr()`

Compatibility Implemented only on devices running Palm OS Garnet or Palm OS Cobalt where the ARM Debug Nub Feature Set is present.

AdnDebugNativeUnregisterAddr Macro

Purpose Ask the debugger to unregister a PACE native object (PNO) identified by its code base address.

Declared In `AdnDebugMgr.h`

Prototype

```
#define AdnDebugNativeUnregisterAddr (_codeAddr)
```

Parameters

→ *_codeAddr*
PNO code base address.

Returns Nothing.

Comments You should call this macro prior to unlocking the PNO code resource so that the debugger breakpoints can be removed.

Generally, PNOs which are unregistering themselves should use the simpler form, `AdnDebugNativeUnregister()`. `AdnDebugNativeUnregisterAddr()` is useful in the case where the code making this call needs to unregister native code that lives elsewhere (for example, in a different chunk), and thus `AdnGetNativeCodeBaseAddr()` won't retrieve the correct code base address.

See Also `AdnDebugNativeRegisterAddr()`

Compatibility Implemented only on devices running Palm OS Garnet or Palm OS Cobalt where the ARM Debug Nub Feature Set is present.

AdnDebugUpdateLoadedModules Macro

Purpose Notify Palm OS Debugger of any recently loaded or unloaded native modules. This call is not used for debugging PACE native objects (PNOs).

Declared In `AdnDebugMgr.h`

Prototype

```
#define AdnDebugUpdateLoadedModules ()
```

Parameters None.

Returns Nothing.

Compatibility Implemented only on devices running Palm OS Garnet or Palm OS Cobalt where the ARM Debug Nub Feature Set is present.

AdnGetNativeCodeBaseAddr Macro

Purpose Obtain the code base address for a PACE native object (PNO).

Declared In `AdnDebugMgr.h`

Prototype

```
#define AdnGetNativeCodeBaseAddr ()
```

Parameters None.

Returns The invoking PNO's code base address.

Comments This macro is used by `AdnDebugNativeRegister()` and `AdnDebugNativeUnregister()` to obtain the PNO's code base address.

Compatibility Implemented only on devices running Palm OS Garnet or Palm OS Cobalt where the ARM Debug Nub Feature Set is present.

SetR10ToDataPointer Macro

Purpose Helper routine that sets register r10 to point to your data section (the GOT from gcc).

Declared In `AdnDebugMgr.h`

Prototype

```
#define SetR10ToDataPointer (ptr)
```

Parameters

→ *ptr*
Pointer to your data section; r10 is set to this value.

Returns Nothing.

Comments This macro is only useful if you have compiled your code to be r10-relative.

Compatibility Implemented only on devices running Palm OS Garnet or Palm OS Cobalt where the ARM Debug Nub Feature Set is present.

26

Bitmaps

This chapter provides information about bitmaps by discussing these topics:

- Bitmap Data Structures
- Bitmap Constants
- Bitmap Resources
- Bitmap Functions

The header file `Bitmap.h` declares the API that this chapter describes. For more information on bitmaps, see the section "Bitmaps" on page 131 in the *Palm OS Programmer's Companion*, vol. I.

Bitmap Data Structures

BitmapCompressionType Enum

Purpose

The `BitmapCompressionType` enum specifies possible bitmap compression types. These are the possible values for the `compressionType` field of `BitmapType`. You can compress or uncompress a bitmap using a call to `BmpCompress()`.

Prototype

```
typedef enum {
  BitmapCompressionTypeScanLine = 0,
  BitmapCompressionTypeRLE,
  BitmapCompressionTypePackBits,
  BitmapCompressionTypeEnd,
  BitmapCompressionTypeBest = 0x64,
  BitmapCompressionTypeNone = 0xFF
} BitmapCompressionType;
```

Constants

`BitmapCompressionTypeScanLine`
: Use scan line compression. Scan line compression is compatible with Palm OS® 2.0 and higher.

`BitmapCompressionTypeRLE`
: Use RLE compression. RLE compression is supported in Palm OS 3.5 and higher.

`BitmapCompressionTypePackBits`
: Use PackBits compression. PackBits compression is supported in Palm OS 4.0 and higher. See `http://partners.adobe.com/asn/developer/pdfs/tn/TIFF6.pdf` for a description of the PackBits format. (Although the TIFF spec is byte-oriented, the 16-bit algorithm is identical: just substitute "word" for "byte.")

`BitmapCompressionTypeEnd`
: For internal use only.

`BitmapCompressionTypeBest`
: For internal use only.

`BitmapCompressionTypeNone`
: No compression is used.

  This value should only be used as an argument to `BmpCompress()`.

Compatibility

`BitmapCompressionType` is only defined if 3.5 New Feature Set is present. Earlier releases do support compressed bitmaps, but in scan line format only.

BitmapDirectInfoType Struct

Purpose

For direct color bitmaps—each pixel is represented by an RGB triplet rather than a palette index—the `BitmapDirectInfoType` structure follows the color table if one is present, or immediately follows the `BitmapType` if a color table is not present. For direct color bitmaps, only 16 bits per pixel is supported, 5 bits for red, 6 bits for green, and 5 bits for blue.

WARNING! This structure is documented so that you can directly access the internals of your own bitmap resources. *Bitmaps created by Palm OS are not guaranteed to adhere to this structure*; you cannot cast a direct color bitmap data pointer from a bitmap created by the Palm OS to this structure and expect to be able to correctly access the structure's fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

Prototype

```
typedef struct BitmapDirectInfoType {
  UInt8 redBits;
  UInt8 greenBits;
  UInt8 blueBits;
  UInt8 reserved;
  RGBColorType transparentColor;
} BitmapDirectInfoType;
```

Fields

`redBits`
: Number of bits used by the red component in each pixel.

`greenBits`
: Number of bits used by the green component in each pixel.

`blueBits`
: Number of bits used by the blue component in each pixel.

`reserved`
: Must be zero. Reserved for future use.

`transparentcolor`
: Contains the red, green, and blue components of the transparent color.

Compatibility

`BitmapDirectInfoType` is only defined if 4.0 New Feature Set is present.

BitmapFlagsType Struct

Purpose

The `BitmapFlagsType` bit field defines the `flags` field of `BitmapType`. It specifies the bitmap's attributes.

WARNING! This structure is documented so that you can directly access the internals of your own bitmap resources. *Bitmaps created by Palm OS are not guaranteed to adhere to this structure*; you cannot cast the `flags` field of a bitmap created by the Palm OS to this structure and expect to be able to correctly access the structure's fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

Prototype

```
typedef struct BitmapFlagsType {
  UInt16 compressed:1;
  UInt16 hasColorTable:1;
  UInt16 hasTransparency:1;
  UInt16 indirect:1;
  UInt16 forScreen:1;
  UInt16 directColor:1;
  UInt16 indirectColorTable:1;
  UInt16 noDither:1;
  UInt16 reserved:8;
} BitmapFlagsType;
```

Fields

`compressed`
: If `true`, the bitmap is compressed and the `compressionType` field specifies the compression used. If `false`, the bitmap is uncompressed. The `BmpCompress()` function sets this field.

`hasColorTable`
: If `true`, the bitmap has its own color table. If `false`, the bitmap uses the system color table. You specify whether the bitmap has its own color table when you create the bitmap.

`hasTransparency`
: If `true`, the OS will not draw pixels that have a value equal to the `transparentIndex`. If `false`, the bitmap has no transparency value. You specify the transparent color when you create the bitmap using Constructor, or you can specify it programmatically with `BmpSetTransparentValue()`. To obtain the value of this field, call `BmpGetTransparentValue()`.

`indirect`
: If `true`, the address to the bitmap's data is stored where the bitmap itself would normally be stored. The actual bitmap data is stored elsewhere. If `false`, the bitmap data is stored directly following the bitmap header or directly following the bitmap's color table if it has one. Never set this flag.

  Note that this flag is supported for bitmaps created by Palm OS only; this flag is not used in user-created bitmap resources.

`forScreen`
: If `true`, bitmap intended for the display (screen) window. Never set this flag.

  Note that this flag is supported for bitmaps created by Palm OS only; this flag is not used in user-created bitmap resources.

`directColor`
: If `true`, bitmap is a direct color (RGB) bitmap.

`indirectColorTable`
: If `true`, and if `hasColorTable` is `true`, a pointer to the bitmap's color table immediately follows the `BitmapType` structure. If `false`, and `hasColorTable` is `true`, the color table immediately follows the `BitmapType` structure. If `hasColorTable` is `false`, `indirectColorTable` is ignored. The `indirect` bit uses similar logic: if both the color table and the bitmap data are indirect, the color table pointer precedes the bitmap data pointer.

  Note that this flag is supported for bitmaps created by Palm OS only; this flag is not used in user-created bitmap resources.

`noDither`
: If `true`, the blitter does not dither the bitmap. If `false`, the source bitmap is dithered if it has a bit depth greater than the destination bitmap.

`reserved`
: Reserved for future use.

Compatibility

The `hasTransparency`, `indirect`, and `forScreen` flags are only defined if 3.5 New Feature Set is present. The `directColor` flag is only defined if 4.0 New Feature Set is present. The

`indirectColorTable` and `noDither` flags are only defined if the High-Density Display Feature Set is present.

BitmapPtr Typedef

Purpose The `BitmapPtr` type defines a pointer to a `BitmapType` structure.

Prototype

```
typedef BitmapType *BitmapPtr;
```

BitmapType Struct

Purpose The `BitmapType` structure represents that which is common to all `BitmapTypeV`*x* structures (`BitmapTypeV0`, `BitmapTypeV1`, `BitmapTypeV2`, and `BitmapTypeV3`). The BitmapType structures define both the bitmaps representing the window display and bitmap resources (`'Tbmp'` and `'tAIB'`) that you create using Constructor or some other application and load into your program.

Because `BitmapType` is merely a portion of the `BitmapTypeV`*x* structures, you should never do `sizeof(BitmapType)`.

WARNING! This structure is documented so that you can directly access the internals of your own bitmap resources. *Bitmaps created by Palm OS are not guaranteed to adhere to this structure*; you cannot cast a bitmap created by the Palm OS to this structure and expect to be able to correctly access the structure's fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

Prototype

```
typedef struct BitmapType {
  Int16 width;
  Int16 height;
  UInt16 rowBytes;
  BitmapFlagsType flags;
  UInt8 pixelSize;
  UInt8 version;
} BitmapType;
typedef BitmapType *BitmapPtr;
```

Fields

`width`
The width of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions()` to access this field.

`height`
The height of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions()` to access this field.

`rowBytes`
The number of bytes stored for each row of the bitmap where height is the number of rows. Use `BmpGetDimensions()` to obtain the contents of this field.

`flags`
The bitmap's attributes. See `BitmapFlagsType`.

`pixelSize`
The pixel depth. Currently supported pixel depths are 1, 2, 4, and 8-bit. You specify this value when you create the bitmap. Use `BmpGetBitDepth()` to access the contents of this field.

`version`
The version of bitmap encoding used. See "Bitmap Constants" on page 546. The value in this field determines the data structure to use when interpreting the fields following `version`: a value of `BitmapVersionZero` (0) corresponds to `BitmapTypeV0`, `BitmapVersionOne` (1) corresponds to `BitmapTypeV1`, `BitmapVersionTwo` (2) corresponds to `BitmapTypeV2`, and `BitmapVersionThree` (3) corresponds to `BitmapTypeV3`. Use `BmpGetVersion()` to obtain the contents of this field.

Comments

Note the following about the `BitmapType` structures:

- None of these fields contains the actual bitmap data. Instead, the bitmap data is stored immediately following the `BitmapTypeV`*x* (which one depends on the value of the `version` field) header structure. If the bitmap has its own color table, the color table is stored in between the header and the data. If the bitmap has a pixel size of 16, and the bitmap is `BitmapTypeV2`, the `BitmapDirectInfoType` structure is stored between the header and the data. You can retrieve a bitmap's data by passing its `BitmapType`

structure to `BmpGetBits`, and you can retrieve its color table with `BmpGetColortable`.

- Unlike most other user interface structures, the `BitmapType` does not store the bitmap's location on the screen. The `WindowType` or the `FormBitmapType` with which this bitmap is associated contains that information.
- A bitmap may be part of a bitmap family. A bitmap family is a group of bitmaps, each containing the same drawing but at a different pixel depth (see "Bitmaps" on page 131 of the *Palm OS Programmer's Companion*, vol. I). When requested to draw a bitmap family, the operating system chooses a member of the bitmap family based upon the bitmap density and pixel depth; see "Bitmap Families" on page 138 for the algorithm that the High-Density Display Feature Set uses to determine which one to choose.

BitmapTypeV0 Struct

Purpose

Structure corresponding to the version 0 encoding of a bitmap. Version 0 encoding is supported in Palm OS 1.0 and later.

Generally you work with pointers to `BitmapType` structures; if the structure's version is `BitmapVersionZero`, the structure is of type `BitmapTypeV0`.

WARNING! This structure is documented so that you can directly access the internals of your own bitmap resources. *Bitmaps created by Palm OS are not guaranteed to adhere to this structure*; you cannot cast a bitmap created by the Palm OS to this structure and expect to be able to directly access the structure's fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

Prototype

```
typedef struct BitmapTypeV0 {
  Int16 width;
  Int16 height;
  UInt16 rowBytes;
  BitmapFlagsType flags;
  UInt16 reserved[4];
} BitmapTypeV0;
```

```
typedef BitmapTypeV0 *BitmapPtrV0;
```

Fields

`width`
: The width of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions()` to access this field.

`height`
: The height of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions()` to access this field.

`rowBytes`
: The number of bytes stored for each row of the bitmap where height is the number of rows. Use `BmpGetDimensions()` to access this field.

`flags`
: The bitmap's attributes. See `BitmapFlagsType`. Only the `compressed` flag is defined for `BitmapTypeV0` structures.

`reserved`
: Reserved. These values are set to zero. Note that in the `BitmapTypeV0` structure, the `pixelSize` and `version` fields, defined in `BitmapType`, do not exist. They coincide with the `reserved` array, however, and this array was initialized to zero when the bitmap was created. The operating system recognizes that a `pixelSize` of zero means that the bitmap's depth is 1.

Compatibility

`BitmapTypeV0` is defined only if High-Density Display Feature Set is present.

BitmapTypeV1 Struct

Purpose

Structure corresponding to the version 1 encoding of a bitmap. Version 1 encoding is supported in Palm OS 3.0 and later.

Generally you work with pointers to `BitmapType` structures; if the structure's version is `BitmapVersionOne`, the structure is of type `BitmapTypeV1`.

WARNING! This structure is documented so that you can directly access the internals of your own bitmap resources. *Bitmaps created by Palm OS are not guaranteed to adhere to this structure*; you cannot cast a bitmap created by the Palm OS to this structure and expect to be able to directly access the structure's fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

Prototype

```
typedef struct BitmapTypeV1 {
  Int16 width;
  Int16 height;
  UInt16 rowBytes;
  BitmapFlagsType flags;
  UInt8 pixelSize;
  UInt8 version;
  UInt16 nextDepthOffset;
  UInt16 reserved[2];
} BitmapTypeV1;
typedef BitmapTypeV1 *BitmapPtrV1;
```

Fields

`width`
: The width of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions()` to access this field.

`height`
: The height of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions()` to access this field.

`rowBytes`
: The number of bytes stored for each row of the bitmap where height is the number of rows. Use `BmpGetDimensions()` to access this field.

`flags`
: The bitmap's attributes. See `BitmapFlagsType`. Only the `compressed` and `hasColorTable` flags are defined for `BitmapTypeV1` structures.

`pixelSize`
: The pixel depth. Currently supported pixel depths are 1, 2, and 4-bit. You specify this value when you create the bitmap. Use `BmpGetBitDepth` to obtain the contents of this field.

`version`
: The version of bitmap encoding used. This field has a value of `BitmapVersionOne` (1) for `BitmapTypeV1` structures. Use `BmpGetVersion()` to obtain the contents of this field.

`nextDepthOffset`
: For bitmap families, this field specifies the start of the next bitmap in the family. The value it contains is the number of 4-byte words to the next `BitmapType` from the beginning of this one. If the bitmap is not part of a bitmap family or it is the last bitmap in the family, the `nextDepthOffset` is 0.

`reserved`
: Reserved.

Compatibility `BitmapTypeV1` is defined only if High-Density Display Feature Set is present.

BitmapTypeV2 Struct

Purpose Structure corresponding to the version 2 encoding of a bitmap. Version 2 encoding is supported in Palm OS 3.5 and later.

Generally you work with pointers to `BitmapType` structures; if the structure's version is `BitmapVersionTwo` (2), the structure is of type `BitmapTypeV2`.

WARNING! This structure is documented so that you can directly access the internals of bitmaps that you create. *Bitmaps created by Palm OS are not guaranteed to adhere to this structure*; you cannot cast a bitmap created by the Palm OS to this structure and expect to be able to directly access the structure's fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

Prototype

```
typedef struct BitmapTypeV2 {
  Int16 width;
  Int16 height;
  UInt16 rowBytes;
  BitmapFlagsType flags;
  UInt8 pixelSize;
  UInt8 version;
```

```
  UInt16 nextDepthOffset;
  UInt8 transparentIndex;
  UInt8 compressionType;
  UInt16 reserved;
} BitmapTypeV2;
typedef BitmapTypeV2 *BitmapPtrV2;
```

Fields

`width`
: The width of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions()` to access this field.

`height`
: The height of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions()` to access this field.

`rowBytes`
: The number of bytes stored for each row of the bitmap where height is the number of rows. Use `BmpGetDimensions()` to access this field.

`flags`
: The bitmap's attributes. See `BitmapFlagsType`. Only the `compressed`, `hasColorTable`, `hasTransparency`, `indirect`, `forScreen`, and `directColor` flags are defined for `BitmapTypeV2` structures. Note that the `indirect and forScreen` flags are system-only flags that are not used in user-created bitmap resources.

`pixelSize`
: The pixel depth. Currently supported pixel depths are 1, 2, 4, 8, and 16-bit. You specify this value when you create the bitmap. Use `BmpGetBitDepth()` to obtain the contents of this field.

`version`
: The version of bitmap encoding used. This field has a value of `BitmapVersionTwo` (2) for `BitmapTypeV2` structures. Use `BmpGetVersion()` to obtain the contents of this field.

`nextDepthOffset`
: For bitmap families, this field specifies the start of the next bitmap in the family. The value it contains is the number of 4-byte words to the next `BitmapType` from the beginning of

this one. If the bitmap is not part of a bitmap family or it is the last bitmap in the family, the `nextDepthOffset` is 0.

`transparentIndex`
: The color index for the transparent color. Only used for version 2 bitmaps and only when the `hasTransparency` flag is set (see `BitmapFlagsType`). You specify this value when you create the bitmap using Constructor, or programmatically with `BmpSetTransparentValue()`. To obtain the value of this field, call `BmpGetTransparentValue()`.

`compressionType`
: The compression type used. Only used for version 2 bitmaps and only when the `compressed` flag is set (see `BitmapFlagsType`). See `BitmapCompressionType` for possible values. The `BmpCompress()` function sets this field, and the `BmpGetCompressionType()` function obtains its value.

`reserved`
: Reserved.

Compatibility

`BitmapTypeV2` is defined only if High-Density Display Feature Set is present.

BitmapTypeV3 Struct

Purpose

Structure corresponding to the version 3 encoding of a bitmap. Version 3 encoding is supported if the High-Density Display Feature Set is present.

Generally you work with pointers to `BitmapType` structures; if the structure's version is `BitmapVersionThree` (3), the structure is of type `BitmapTypeV3`.

`BmpCreate` allocates and initializes a `BitmapTypeV2` structure. To create a `BitmapTypeV3` structure, use `BmpCreateBitmapV3()` and supply the data pointer and optional color table pointer.

In earlier versions of the `BitmapTypeV`*x* structure, the size of compressed bitmap data is stored in a 16-bit field preceding the bitmap data. With the version 3 structure, the size is stored in a 32-bit field.

The `BitmapTypeV3` structure has fields that identify how each pixel is stored (`pixelFormat`) and which color, if any, is "transparent" (`transparentValue`). Because of this, you don't use a `BitmapDirectInfoType` structure in conjunction with a `BitmapTypeV3` structure.

WARNING! This structure is documented so that you can directly access the internals of bitmaps that you create. *Bitmaps created by Palm OS are not guaranteed to adhere to this structure*; you cannot cast a bitmap created by the Palm OS to this structure and expect to be able to directly access the structure's fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

Prototype

```
typedef struct BitmapTypeV3 {
  Int16 width;
  Int16 height;
  UInt16 rowBytes;
  BitmapFlagsType flags;
  UInt8 pixelSize;
  UInt8 version;
  UInt8 size;
  UInt8 pixelFormat;
  UInt8 unused;
  UInt8 compressionType;
  UInt16 density;
  UInt32 transparentValue;
  UInt32 nextBitmapOffset;
} BitmapTypeV3;
typedef BitmapTypeV3 *BitmapPtrV3;
```

Fields

`width`
: The width of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions()` to access this field.

`height`
: The height of the bitmap in pixels. You specify this value when you create the bitmap. Use `BmpGetDimensions()` to access this field.

`rowBytes`
: The number of bytes stored for each row of the bitmap where height is the number of rows. Use `BmpGetDimensions()` to access this field.

`flags`
: The bitmap's attributes. See `BitmapFlagsType`. Note that the `indirect`, `forScreen`, and `indirectColorTable` flags are system-only fields that are not used in user-created bitmap resources.

`pixelSize`
: The pixel depth. Currently supported pixel depths are 1, 2, 4, and 8-bit. You specify this value when you create the bitmap. Use `BmpGetBitDepth()` to obtain the value of this field.

`version`
: The version of bitmap encoding used. This field has a value of `BitmapVersionThree` (3) for `BitmapTypeV3` structures. The high bit of the version field is set if the bitmap data structure uses the native ARM format, with little-endian fields. `Bitmap.h` contains a bit mask, `BitmapVersionMaskLE`, that can be used to detect this. Use `BmpGetVersion()` to obtain the value of this field.

`size`
: The size of this structure, in bytes. This field does not include the size of the color table or the size of the bitmap data. Use `BmpGetSizes()` to obtain the value of this field.

`pixelFormat`
: An enumerated constant representing the format of the pixel data. See `PixelFormatType` for the supported values.

`unused`
: Not used.

`compressionType`
: The compression type used; 0 if the bitmap is not compressed. Only used when the `compressed` flag is set (see `BitmapFlagsType`). See `BitmapCompressionType` for possible values. The `BmpCompress()` function sets this field, and the `BmpGetCompressionType()` function obtains its value.

`density`
: Value used by the blitter to determine how to stretch or shrink the bitmap data. For the screen bitmap, this field represents the screen density. For handhelds with low-density displays, this field is initialized to `kDensityLow`. For handhelds with double-density displays, this field is initialized to `kDensityDouble`. See `DensityType` for the full set of values that this field can assume. Set this field with `BmpSetDensity()`, and obtain its value with `BmpGetDensity()`.

`transparentValue`
: If this structure represents a bitmap with a bit depth of 8 or less, this field contains the bitmap's transparent index. If the bitmap has a bit depth of 16, the 16-bit transparent RGB color is stored in this field. You specify this value when you create the bitmap using Constructor, or programmatically with `BmpSetTransparentValue()` To obtain the value of this field, call `BmpGetTransparentValue()`.

`nextBitmapOffset`
: A 32-bit value that indicates the number of bytes to the next bitmap in the family. If the bitmap is not part of a bitmap family or it is the last bitmap in the family, the `nextBitmapOffset` is 0.

Compatibility `BitmapTypeV3` is defined only if High-Density Display Feature Set is present.

ColorTableType Struct

Purpose The `ColorTableType` structure defines a color table. Bitmaps can have color tables attached to them; however, doing so is not recommended for performance reasons.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `ColorTableType` structure. Never access its structure members directly, or your code may break in future versions. Use `BmpGetColortable()` to access this structure. Use the information below for debugging purposes only.

Prototype

```
typedef struct ColorTableType {
  UInt16 numEntries;
  // RGBColorType entry[];
} ColorTableType;
```

Fields

`numEntries`
: The number of entries in table. High bits (`numEntries` > 256) reserved.

The color table entries themselves are of type `RGBColorType`, and there is one per `numEntries`. Use the macro `ColorTableEntries()` to retrieve these entries.

Care should be taken not to confuse a full color table (which includes the count) with an array of RGB color values. Some routines operate on entire color tables; others operate on lists of color entries.

Compatibility

`ColorTableType` is defined only if 3.5 New Feature Set is present.

DensityType Enum

Purpose

The density of the bitmap (see "Display Density" on page 80 of the *Palm OS Programmer's Companion*, vol. I for a definition of display density). Density is only supported in `BitmapType` structures with a version greater than 2; if a given `BitmapType` structure is version 2 or lower, it is assumed to contain low-density data.

The blitter uses the density field in the source and destination bitmaps to determine an appropriate scaling factor. When scaling down from a density of `kDensityDouble` to `kDensityLow`, the blitter must shrink the bitmap data. This will almost always result in a poorer-quality image when compared with a bitmap that was created with a density of `kDensityLow`.

The various `DensityType` values should not be interpreted as representing pixels per inch.

Prototype

```
typedef enum {
  kDensityLow = 72,
  kDensityOneAndAHalf = 108,
  kDensityDouble = 144,
  kDensityTriple = 216,
  kDensityQuadruple = 288
} DensityType
```

Constants

`kDensityLow`
: Low (single) density. A low-density screen is 160x160 pixels.

`kDensityOneAndAHalf`
: "One and a half" density. A one-and-a-half-density display is 240x240 pixels; this would most likely be used on a handheld with a 240x320 screen where the bottom portion is used as a dynamic input area.

`kDensityDouble`
: Double density when compared with `kDensityLow`. A double-density screen is 320x320 pixels.

`kDensityTriple`
: Triple density when compared with `kDensityLow`. A triple-density screen is 480x480 pixels.

`kDensityQuadruple`
: Quadruple density when compared with `kDensityLow`. A quadruple-density screen is 640x640 pixels.

IMPORTANT: Not all densities listed in the `DensityType` enum are supported by a given version of the High-Density Display feature set. For Palm OS Garnet, only `kDensityLow`, `kDensityOneAndAHalf`, and `kDensityDouble` are supported.

Compatibility

`DensityType` is defined only if High-Density Display Feature Set is present.

PixelFormatType Enum

Purpose

Pixel formats defined for use with `BitmapTypeV3` structures.

Prototype

```
typedef enum {
  pixelFormatIndexed,
  pixelFormat565
  pixelFormat565LE,
  pixelFormatIndexedLE
} PixelFormatType;
```

Constants

`pixelFormatIndexed`
: Each pixel is represented by a palette index.

`pixelFormat565`
: Each pixel is represented by an RGB triplet stored in 16-bits: 5 red bits, 6 green bits, and 5 blue bits.

`pixelFormat565LE`
: Similar to `pixelFormat565`, except that the 16 bits of the RGB triplet are stored as little-endian. This pixel format is not supported in user-created bitmaps.

`pixelFormatIndexedLE`
: Similar to `pixelFormatIndexed`, except that the pixels within a byte are stored as little-endian. This pixel format is not supported in user-created bitmaps.

Compatibility

`PixelFormatType` is defined only if High-Density Display Feature Set is present.

RGBColorType Struct

Purpose

The `RGBColorType` structure defines a color. It is used as an entry in the color table. `RGBColorTypes` can also be created manually and passed to several user interface functions.

Prototype

```
typedef struct RGBColorType {
  UInt8 index;
  UInt8 r;
  UInt8 g;
  UInt8 b;
} RGBColorType;
```

Fields

`index`
: The index of this color in the color table. Not all functions that use `RGBColorType` use the `index` field.

  Direct bitmaps support no more than 256 colors. The number of possible RGB colors greatly exceeds this amount. For this

reason, some drawing functions use a color look up table (CLUT). If the CLUT is used, the `index` field contains the index of an available color that is the closest match to the color specified by the `r`, `g`, and `b` fields.

`r`
: Amount of red (0 to 255).

`g`
: Amount of green (0 to 255).

`b`
: Amount of blue (0 to 255).

Compatibility `RGBColorType` is defined only if 3.5 New Feature Set is present.

Bitmap Constants

`BitmapVersionZero = 0`
: Uses the version 0 encoding of a bitmap. Version 0 encoding is supported in Palm OS 1.0 and later.

`BitmapVersionOne = 1`
: Uses the version 1 encoding of a bitmap. Version 1 encoding is supported in Palm OS 3.0 and later.

PalmRez automatically creates version 1 bitmaps unless you have specified a transparency index or a compressed type when creating the bitmap in Constructor.

`BitmapVersionTwo = 2`
: Uses the version 2 encoding of a bitmap. Palm OS 3.5 and later supports version 2 bitmaps. Version 2 bitmaps either use the transparency index or are compressed. If you programmatically create a bitmap using `BmpCreate()`, a version 2 bitmap is created.

`BitmapVersionThree = 3`
: Uses the version 3 encoding of a bitmap. Version 3 bitmaps are supported only if the High-Density Display Feature Set is present.

Bitmap Resources

You can create a bitmap resource and include it as part of your application's PRC file. Use the resource type `'Tbmp'` for most images and the resource type `'tAIB'` for application icons. Symbolically, these two resource types are `bitmapRsc` and `iconType`, respectively.

Note that if you are creating a bitmap or a bitmap family in Constructor, you create a `'tbmf'` resource (or `'taif'` resource for icons) and one or more 'PICT' images. The PalmRez post linker converts them into a single `'Tbmp'` or `'tAIB'` resource. Note that the PalmRez post linker takes PICT images even on the Microsoft Windows operating system.

Bitmap Functions

BmpBitsSize Function

Purpose Return the size of the bitmap's data.

Declared In `Bitmap.h`

Prototype

```
UInt16 BmpBitsSize (const BitmapType *bitmapP)
```

Parameters

→ *bitmapP*
Pointer to the bitmap. (See `BitmapType`.)

Returns Returns the size in bytes of the bitmap's data, excluding the header and the color table.

Comments This function returns the bitmap's data size even if the bitmap's `indirect` flag is set. (See `BitmapFlagsType`.)

If the bitmap is compressed, this function returns the compressed size of the bitmap.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `BmpSize()`, `BmpColortableSize()`, `BmpGetBits()`, `BmpGetSizes()`

BmpColortableSize Function

Purpose Return the size of the bitmap's color table.

Declared In `Bitmap.h`

Prototype

```
UInt16 BmpColortableSize
    (const BitmapType *bitmapP)
```

Parameters → *bitmapP*
Pointer to the bitmap. (See `BitmapType`.)

Returns Returns the size in bytes of the bitmap's color table or 0 if the bitmap does not use its own color table.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `BmpBitsSize()`, `BmpSize()`, `BmpGetColortable()`

BmpCompress Function

Purpose Compress or uncompress a bitmap.

Declared In `Bitmap.h`

Prototype

```
Err BmpCompress (BitmapType *bitmapP,
    BitmapCompressionType compType)
```

Parameters → *bitmapP*
Pointer to the bitmap to compress. (See `BitmapType`.)

→ *compType*
The type of compression to use. (See `BitmapCompressionType`.) If set to `BitmapCompressionTypeNone` and `bitmapP` is compressed, this function uncompresses the bitmap.

Returns Returns one of the following values:

`errNone`
Success.

`sysErrParamErr`
Either the `compType` parameter does not specify a compression type or the bitmap is already compressed, is in the storage heap, or represents the screen.

`sysErrNoFreeResource`
: There is not enough memory available to complete the operation.

Comments This function performs the specified compression and resizes the bitmap's allocated memory. The bitmap must be in the dynamic heap.

Compatibility Implemented only if 3.5 New Feature Set is present. From Palm OS Garnet version 5.0 onwards this function does nothing; `errNone` is always returned.

See Also `BmpGetCompressionType()`

BmpCreate Function

Purpose Create a bitmap.

Declared In `Bitmap.h`

Prototype

```
BitmapType *BmpCreate (Coord width, Coord height,
   UInt8 depth, ColorTableType *colortableP,
   UInt16 *error)
```

Parameters

→ *width*
: The width of the bitmap in pixels. Must not be 0.

→ *height*
: The height of the bitmap in pixels. Must not be 0.

→ *depth*
: The pixel depth of the bitmap. Must be 1, 2, 4, 8, or 16. This value is used as the `pixelSize` field of `BitmapType`.

→ *colortableP*
: A pointer to the color table associated with the bitmap, or `NULL` if the bitmap should not include a color table. If specified, the number of colors in the color table must match the `depth` parameter. (2 for 1-bit, 4 for 2-bit, 16 for 4-bit, and 256 for 8-bit). 16-bit bitmaps do not use a color table.

← *error*
: Contains the error code if an error occurs.

Returns Returns a pointer to the new bitmap structure (see `BitmapType`) or `NULL` if an error occurs. The parameter `error` contains one of the following:

`errNone`
: Success.

`sysErrParamErr`
: The `width`, `height`, `depth`, or `colorTableP` parameter is invalid. See the descriptions above for acceptable values.

`sysErrNoFreeResource`
: There is not enough memory available to allocate the structure.

Comments

This function creates an uncompressed, non-transparent `BitmapVersionTwo` bitmap with the width, height, and depth that you specify. To create a`BitmapVersionThree` bitmap use `BmpCreate` and pass the results to `BmpCreateBitmapV3()`.

If you pass a color table, the bitmap's `hasColorTable` flag is set. For performance reasons, attaching a custom color table to a bitmap is strongly discouraged. An alternative is to use the `WinPalette()` command to change the color table as needed, draw the bitmap, and then undo your changes after you have finished displaying the bitmap.

`BmpCreate` allocates sufficient memory on the dynamic heap to hold the bitmap and initializes all of its pixels to white. To change the bitmap's contents, use the window drawing functions. First, you must use `WinCreateBitmapWindow()` to create an off screen window wrapper around the bitmap, then draw to that window. For example:

```
BitmapType *bmpP;
WinHandle win;
Err error;
RectangleType onScreenRect;

bmpP = BmpCreate(10, 10, 8, NULL, &error);
if (bmpP) {
  win = WinCreateBitmapWindow(bmpP, &error);
  if (win) {
    WinSetDrawWindow(win);
    WinDrawLines(win, ...);
    /* etc */
    WinSetWindowBounds(win, onScreenRect);
  }
}
```

You cannot use this function to create a bitmap written directly to a database; that is, you must create the bitmap on the dynamic heap first, then write it to the storage heap.

It is not necessary to use `BmpCreate` to load a bitmap stored in a resource. Instead, you simply load the resource and lock its handle. The returned pointer is a pointer to a `BitmapType`. For example:

```
MemHandle resH =
  DmGetResource (bitmapRsc, rscID);
BitmapType *bitmap = MemHandleLock (resH);
```

Bitmaps 64 Kb and greater are now supported with Palm OS 4.0.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `BmpCreateBitmapV3()`, `BmpDelete()`

BmpCreateBitmapV3 Function

Purpose Create a version 3 bitmap from an existing bitmap, an existing set of data bits, and, optionally, a color table.

Declared In `Bitmap.h`

Prototype

```
BitmapTypeV3 *BmpCreateBitmapV3
    (const BitmapType *bitmapP, UInt16 density,
    const void *bitsP,
    const ColorTableType *colorTableP)
```

Parameters

→ *bitmapP*
Pointer to a valid bitmap from which the version 3 bitmap is to be created. See `BitmapType`.

→ *density*
Density of the returned bitmap. If 0, the returned bitmap's density is set to the default value of `kDensityLow`.

→ *bitsP*
Pointer to the bitmap image data. Note that the bitmap data can be located in the storage heap, but then the bitmap should be treated as read-only. You must use `DmWrite` to write to the storage heap; blitting to it causes a system error.

→ *colorTableP*
Pointer to a color table, or `NULL` to use `bitmapP`'s color table, if one exists.

Returns Returns a version 3 bitmap, or `NULL` if the bitmap could not be created from the specified bitmap, bitmap data, and optional color table.

Comments You can use this function when the bitmap data is stored in the storage heap as bands of raster data. Rather than allocating several bitmap structures, one for each band, use this function to allocate a single bitmap, and have the structure point to each band successively. This is typically used with high-density bitmaps that cannot be stored entirely within 64k.

WARNING! Due to a limitation in the way that this function is implemented, `BitmapCreateBitmapV3` doesn't work with compressed bitmaps. Don't pass bitmaps to this function that have the `compressed` flag set.

The returned bitmap structure is allocated from the system heap. After your application is done with it, dispose of it by calling `BmpDelete()`.

Compatibility Implemented only if the High-Density Display Feature Set is present.

See Also `BmpCreate()`

BmpDelete Function

Purpose Delete a bitmap structure.

Declared In `Bitmap.h`

Prototype

```
Err BmpDelete (BitmapType *bitmapP)
```

Parameters → *bitmapP*
Pointer to the structure of the bitmap to be deleted. (See `BitmapType`.)

Returns Returns `errNone` upon success, `sysErrParamErr` if the bitmap's `forScreen` flag is set or the bitmap resides in the storage heap. Returns one of the memory errors if the freeing pointer fails.

Comments Only delete bitmaps that have been created using `BmpCreate()`.

You cannot use this function on a bitmap located in a database. To delete a bitmap from a database, use the standard data manager calls.

Compatibility Implemented only if 3.5 New Feature Set is present.

BmpGetBits Function

Purpose Retrieve the bitmap's data.

Declared In `Bitmap.h`

Prototype

```
void *BmpGetBits (BitmapType *bitmapP)
```

Parameters → *bitmapP*
Pointer to the bitmap's structure. (See `BitmapType`.)

Returns Returns a pointer to the bitmap's data.

Comments This function returns the bitmap's data even if the bitmap's `indirect` flag is set. (See `BitmapFlagsType`.)

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `BmpBitsSize()`, `BmpGlueGetBits()`

BmpGetBitDepth Function

Purpose Retrieve the depth of a bitmap.

Declared In `Bitmap.h`

Prototype

```
UInt8 BmpGetBitDepth (const BitmapType *bitmapP)
```

Parameters → *bitmapP*
Pointer to a bitmap. See `BitmapType`.

Returns This function returns the bit depth of the bitmap, as represented by the `pixelSize` field in `BitmapType`. For debug ROMs, this function reports an error and returns 0 if `bitmapP` is `NULL`.

Compatibility Implemented only if 4.0 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS,

link with the PalmOSGlue library and call `BmpGlueGetBitDepth`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `BmpGetDimensions()`, `BmpGetNextBitmap()`, `BmpGetSizes()`

BmpGetColortable Function

Purpose Retrieve the bitmap's color table.

Declared In `Bitmap.h`

Prototype

```
ColorTableType *BmpGetColortable
    (BitmapType *bitmapP)
```

Parameters → *bitmapP*
A pointer to the bitmap. See `BitmapType`.

Returns Returns a pointer to the color table or `NULL` if the bitmap uses the system color table.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `BmpColortableSize()`

BmpGetCompressionType Function

Purpose Get the compression type of a bitmap.

Declared In `Bitmap.h`

Prototype

```
BitmapCompressionType BmpGetCompressionType
    (const BitmapType *bitmapP)
```

Parameters → *bitmapP*
Pointer to a valid bitmap. See `BitmapType`.

Returns Returns the type of compression used by `bitmapP`. See "BitmapCompressionType" on page 527 for the values that can be returned from this function. For debug ROMs, this function reports an error and returns `BitmapCompressionTypeNone` if `bitmapP` is `NULL`.

Comments If the bitmap is not compressed, this function returns `BitmapCompressionTypeNone`. If the bitmap version is 0 or 1

(corresponding to BitmapTypeV0 and BitmapTypeV1, respectively), it returns BitmapCompressionTypeScanLine.

Compatibility Implemented only if either the if 5.0 New Feature Set or the High-Density Display Feature Set is present.

See Also BmpCompress()

BmpGetDensity Function

Purpose Get the density of a bitmap.

Declared In Bitmap.h

Prototype

```
UInt16 BmpGetDensity (const BitmapType *bitmapP)
```

Parameters → *bitmapP*
Pointer to a valid bitmap. See BitmapType.

Returns Returns the density of bitmapP; see the DensityType enum for the defined set of density values. For debug ROMs, this function reports an error and returns 0 if bitmapP is NULL.

Comments Note that bitmaps with a version of 0, 1, or 2 (corresponding to BitmapTypeV0, BitmapTypeV1, and BitmapTypeV2, respectively) are assumed to be low density (kDensityLow).

Compatibility Implemented only if the High-Density Display Feature Set is present.

See Also BmpCreateBitmapV3(), BmpSetDensity()

BmpGetDimensions Function

Purpose Retrieve the width, height and number of data bytes per row of a bitmap.

Declared In Bitmap.h

Prototype

```
void BmpGetDimensions (const BitmapType *bitmapP,
    Coord *widthP, Coord *heightP,
    UInt16 *rowBytesP)
```

Parameters → *bitmapP*
Pointer to the bitmap. See BitmapType.

← *widthP*
Pointer to bitmap's width in pixels. Use `NULL` if this information is not wanted.

← *heightP*
Pointer to bitmap's height in pixels. Use `NULL` if this information is not wanted. Use `NULL` if this information is not wanted.

← *rowBytesP*
Pointer to number of bytes per row of bitmap. Use `NULL` if this information is not wanted.

Returns This function returns the width in pixels of the bitmap in `widthP`, the height in pixels of the bitmap in `heightP`, and the number of bytes of data per row of the bitmap in `rowBytesP`. This function reports an error on debug ROMs if `bitmapP` is `NULL`.

Compatibility Implemented only if 4.0 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `BmpGlueGetDimensions`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `BmpGetBitDepth()`, `BmpGetNextBitmap()`, `BmpGetSizes()`

BmpGetNextBitmap Function

Purpose Retrieve the next low-density bitmap in a bitmap family.

Declared In `Bitmap.h`

Prototype

```
BitmapType *BmpGetNextBitmap(BitmapType *bitmapP)
```

Parameters → *bitmapP*
Pointer to a bitmap. See `BitmapType`.

Returns This function returns a pointer to the next low-density `BitmapType` in a bitmap family. It returns `NULL` if `bitmapP` is the last low-density bitmap. For debug ROMs, this function reports an error and returns 0 if `bitmapP` is `NULL`.

Comments This function is only supported for low-density bitmap families. For families with both low and high-density bitmaps (`BitmapTypeV3`), use `BmpGetNextBitmapAnyDensity()`.

In a `BitmapTypeV3`, a dummy low-density `BitmapType` precedes any high-density bitmaps. If you pass a pointer to the last valid low-density bitmap within a `BitmapTypeV3` structure, `BmpGetNextBitmap` returns a pointer to the dummy bitmap. If you then pass that pointer to `BmpGetDimensions()`, it returns the dimensions for the first high-density bitmap rather than the dummy bitmap. To avoid confusion, it is best to use `BmpGetNextBitmapAnyDensity` when iterating through a family and gathering information about each bitmap.

Compatibility Implemented only if 4.0 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `BmpGlueGetNextBitmap`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `BmpGetBitDepth()`, `BmpGetDimensions()`, `BmpGetSizes()`

BmpGetNextBitmapAnyDensity Function

Purpose Get the next bitmap in the bitmap family, irrespective of density.

Declared In `Bitmap.h`

Prototype

```
BitmapType *BmpGetNextBitmapAnyDensity
    (BitmapType *bitmapP)
```

Parameters

→ *bitmapP*
Pointer to a valid bitmap. See `BitmapType`.

Returns Returns the next bitmap in a bitmap family, or `NULL` if `bitmapP` is the last bitmap. For debug ROMs, this function reports an error and returns 0 if `bitmapP` is `NULL`.

Comments This function is an extended version of `BmpGetNextBitmap()`. For backward compatibility, `BmpGetNextBitmap` only returns low-density bitmaps. If the bitmap family contains high-density bitmaps, however, `BitmapGetNextBitmapAnyDensity` skips over the dummy bitmap that separates the low and high-density bitmaps in the linked list and returns a high-density bitmap.

Compatibility Implemented only if the High-Density Display Feature Set is present.

See Also `BmpGetDensity()`, `BmpGetNextBitmap()`

BmpGetSizes Function

Purpose Retrieve the size of a bitmap and its header structure.

Declared In `Bitmap.h`

Prototype

```
void BmpGetSizes (const BitmapType *bitmapP,
   UInt32 *dataSizeP, UInt32 *headerSizeP)
```

Parameters

→ *bitmapP*
Pointer to the bitmap. See `BitmapType`.

← *dataSizeP*
Pointer to size of bitmap data, not including structures. Use `NULL` if this information is not wanted.

← *headerSizeP*
Pointer to size of bitmap's structures, not including data. Use `NULL` if this information is not wanted.

Returns Returns the size of the bitmap and the size of the bitmap's structures. This function will report an error on debug ROMs if `bitmapP` is `NULL`.

Comments This function returns the size in bytes of the bitmap data in `dataSizeP`. The size does not include the data structures (`BitmapType`, `BitmapDirectInfoType`, or color table) that are associated with a bitmap. The size of the structures (in bytes) are returned in `headerSizeP`, which includes the size of the `BitmapType`, `BitmapDirectInfoType` (if any), the color table (if any), and the size of the pointer for indirect bitmaps (described in `BitmapFlagsType`).

This function should be used when working with bitmaps that may be 64 Kb or greater. Do not use `BmpSize()` or `BmpBitsSize()` when working with bitmaps that may be greater than or equal to 64Kb.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `BmpGetBitDepth()`, `BmpGetDimensions()`, `BmpGetNextBitmap()`

BmpGetTransparentValue Function

Purpose Get a bitmap's transparent color.

Declared In `Bitmap.h`

Prototype

```
Boolean BmpGetTransparentValue
    (const BitmapType *bitmapP,
    UInt32 *transparentValueP)
```

Parameters

→ *bitmapP*
Pointer to a valid bitmap. See `BitmapType`.

← *transparentValueP*
Pointer to a variable that receives the transparent color, either as a palette index or as a direct color value (an RGB565 value), depending on the bitmap's depth.

Returns Returns `true` if `bitmapP` has a transparent color defined, `false` otherwise.

Compatibility Implemented only if either the if 5.0 New Feature Set or the High-Density Display Feature Set is present.

See Also `BmpSetTransparentValue()`

BmpGetVersion Function

Purpose Get the version of a bitmap.

Declared In `Bitmap.h`

Prototype

```
UInt8 BmpGetVersion (const BitmapType *bitmapP)
```

Parameters

→ *bitmapP*
Pointer to a valid bitmap. See `BitmapType`.

Returns Returns the version of `bitmapP`. See"Bitmap Constants" on page 546 for the defined bitmap version numbers. For debug ROMs, this function reports an error and returns 0 if `bitmapP` is `NULL`.

Compatibility Implemented only if the High-Density Display Feature Set is present.

BmpSetDensity Function

Purpose Set the density of a version 3 bitmap.

Declared In `Bitmap.h`

Prototype

```
Err BmpSetDensity (BitmapType *bitmapP,
    UInt16 density)
```

Parameters

→ *bitmapP*
Pointer to a valid version 3 bitmap. See `BitmapTypeV3`.

→ *density*
The bitmap's density. This value should be one of the values defined by the `DensityType` enum.

Returns Returns `errNone` if the operation completed successfully, or `SysErrParamErr` either if `bitmapP` is `NULL`, if `density` is not supported by the blitter, or if `*bitmapP` is not a version 3 bitmap.

Comments To allocate a high-density bitmap, first call `BmpCreateBitmapV3()`. Then call `BmpSetDensity` to specify the bitmap's density.

Compatibility Implemented only if the High-Density Display Feature Set is present.

See Also `BmpGetDensity()`

BmpSetTransparentValue Function

Purpose Set a bitmap's transparent color.

Declared In `Bitmap.h`

Prototype

```
void BmpSetTransparentValue (BitmapType *bitmapP,
    UInt32 transparentValue)
```

Parameters

→ *bitmapP*
Pointer to a valid bitmap. See `BitmapType`.

→ *transparentValue*
Transparent color. This should either be a palette index or a direct color value (an RGB565 value), depending on the bitmap's depth.

Returns Returns nothing.

Comments If `bitmapP` points to a version 2 bitmap, `BmpSetTransparentValue` sets the `BitmapTypeV2` structure's `hasTransparency` flag to `true` and initializes the structure's `transparentIndex` field according to `transparentValue`. For 16-bit bitmaps, this function sets the `transparentColor` field in the `BitmapDirectInfoType` auxiliary structure and sets the `transparentIndex` field to 0.

If `bitmapP` points to a version 3 bitmap, `BmpSetTransparentValue` sets the `BitmapTypeV3` structure's `hasTransparency` flag to `true` and sets the `transparentValue` field to the transparent color.

Regardless of the bitmap version, if this function is passed a `transparentValue` set to `kTransparencyNone`, this function sets the bitmap structure's `hasTransparency` flag to `false` and sets the transparent color field(s) to 0.

This function does nothing if `transparentValue` contains a value that is not valid for the depth of `bitmapP`.

Compatibility Implemented only if either the if 5.0 New Feature Set or the High-Density Display Feature Set is present.

See Also `BmpGetTransparentValue()`

BmpSize Function

Purpose Return the size of the bitmap.

Declared In `Bitmap.h`

Prototype

```
UInt16 BmpSize (const BitmapType *bitmapP)
```

Parameters → *bitmapP*
A pointer to the bitmap. See `BitmapType`.

Returns Returns the size in bytes of the bitmap, including its header, color table (if any), and `sizeof(BitmapDirectInfoType)` if one exists.

Comments If the bitmap has its `indirect` flag set (see `BitmapFlagsType`), the bitmap data is not included in the size returned by this function.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also BmpBitsSize(), BmpColortableSize(), BmpGetSizes()

ColorTableEntries Macro

Purpose Returns the color table.

Declared In Bitmap.h

Prototype

```
#define ColorTableEntries (ctP)
```

Parameters

→ *ctP*
A pointer to a ColorTableType structure.

Returns Returns an array of RGBColorType structures, one for each entry in the color table.

Comments You can use this macro to retrieve the RGB values in use by a bitmap. For example:

```
BitmapType *bmpP;
RGBColorType *tableP =
  ColorTableEntries(BmpGetColorTable(bmpP));
```

If you want to retrieve the RGB values in use by the system color table, you can simply use the WinPalette() function instead of this macro:

```
RGBColorType table[256];
Err e;

/* allocate space for table */
e = WinPalette(winPaletteGet, 0, 256, tableP);
```

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also BmpGetColortable()

27

Byte Order Utilities

These utility macros are useful when passing data back and forth between ARM code (for instance, a PNO) and 68K code.

The header file `ByteOrderUtils.h` declares the API that this chapter describes.

ByteOrderUtils Functions and Macros

EndianSwap16 Macro

Purpose Byte-swap a 16-bit quantity, changing it from big-endian to little-endian or vice-versa.

Declared In `ByteOrderUtils.h`

Prototype

```
#define EndianSwap16 (n)
```

Parameters → *n*

The 16-bit quantity to be byte-swapped.

Returns The byte-swapped equivalent of the supplied 16-bit value.

See Also `EndianSwap32()`

EndianSwap32 Macro

Purpose Byte-swap a 32-bit quantity, changing it from big-endian to little-endian or vice-versa.

Declared In `ByteOrderUtils.h`

Prototype

```
#define EndianSwap32 (n)
```

Parameters → *n*

The 32-bit quantity to be byte-swapped.

Returns The byte-swapped equivalent of the supplied 32-bit value.

See Also `EndianSwap16()`

Read68KUnaligned32 Macro

Purpose Read a 32-bit quantity from a local variable while byte-swapping it. The address from which the 32-bit quantity is read doesn't have to be aligned on either a 2-byte or 4-byte boundary.

Declared In `ByteOrderUtils.h`

Prototype

```
#define Read68KUnaligned32 (addr)
```

Parameters → *addr*
Address from which the value is to be read.

Returns The byte-swapped equivalent of the 32-bit value read from the supplied address.

Comments ARM processors require that 4-byte integers be aligned on a 4-byte boundary. 68K processors require only even address (2-byte) alignment.

See Also `EndianSwap32()`, `Write68KUnaligned32()`

Write68KUnaligned32 Macro

Purpose Write a 32-bit quantity to a local variable while byte-swapping it. The address to which the 32-bit quantity is written doesn't have to be aligned on either a 2-byte or 4-byte boundary.

Declared In `ByteOrderUtils.h`

Prototype

```
#define Write68KUnaligned32 (addr, value)
```

Parameters → *addr*
Address to which the value is to be written.

→ *value*
The 32-bit value to be byte-swapped and written.

Returns Nothing useful.

Comments ARM processors require that 4-byte integers be aligned on a 4-byte boundary. 68K processors require only even address (2-byte) alignment.

See Also `EndianSwap32()`, `Read68KUnaligned32()`

28

Certificate Manager

The Certificate Manager manages digital certificates, handling such operations as import, export, parsing, secure storage, content authentication, and storage querying. You can use the Certificate Manager in two different ways: as a certificate verifier and parser, and as a certificate store. In the verifier/parser mode, the Certificate Manager takes data as input and parses it as a digital certificate. The user can then verify the certificate and access its internal fields. In certificate store mode, the Certificate Manager can securely store a tree of digital certificates (with multiple roots) and make the fields of those certificates available to users.

The Certificate Manager is a system server with a client-side library. To securely store certificates, the Certificate Manager makes use of the Data Manager's vault facilities.

The remainder of this chapter documents the Certificate Manager APIs. It is organized into the following sections:

Certificate Manager Structures and Types 572

Certificate Manager Constants 576

Certificate Manager Element Field Macros 584

Certificate Manager Functions and Macros 586

The header file `CertificateMgr.h` declares the API that this chapter describes.

Certificate Manager Structures and Types

CertMgrCertChainType Struct

Purpose A certificate chain is used when calling `CertMgrVerifyCert()`. You may define a list of certificates to be used when trying to verify the certificate chain.

Declared In `CertificateMgr.h`

Prototype

```
typedef struct {
   CertMgrCertInfoType *certs;
   uint32_t count;
} CertMgrCertChainType
```

Fields

`certs`
: Pointer to the first certificate in the chain.

`count`
: Number of certificates in the chain.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

CertMgrCertElementEnum Typedef

Purpose Certificate element types, used in conjunction with `CertMgrGetField()`.

Declared In `CertificateMgr.h`

Prototype

```
typedef uint32_t CertMgrCertElementEnum
```

Constants

```
#define apCertMgrElementTypeRDN 34
#define apCertMgrElementTypeRSA 33
#define apCertMgrElementTypeX509Cert 32
#define apCertMgrElementTypeX509Extensions 35
```

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

CertMgrCertFieldEnum Typedef

Purpose Certificate element field types, used in conjunction with `CertMgrGetField()`.

Declared In `CertificateMgr.h`

Prototype

```
typedef uint32_t CertMgrCertFieldEnum
```

Comments The values used with variables of this type depend on the certificate element. Depending on the certificate element, the element's field types are listed under one of the following:

- "X509Cert Element Fields" on page 576
- "RSA Element Fields" on page 578
- "RDN Element Fields" on page 579
- "X509Extensions Element Fields" on page 579

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

CertMgrCertInfoType Struct

Purpose Abstracts a certificate object. An application uses a structure of this type to refer to a certificate in the memory address space of the Certificate Manager. You can get a reference to a certificate from a successful call to `CertMgrImportCert()` or `CertMgrFindCert()`

Declared In `CertificateMgr.h`

Prototype

```
typedef struct {
   uint32_t ref;
   uint16_t format;
} CertMgrCertInfoType
```

Fields

`ref`
: An opaque object.

`format`
: The format of the certificate. See "Certificate Formats" on page 581 for the defined set of certificate format values.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

CertMgrCertSearchEnum Typedef

Purpose Specifies the search mode to `CertMgrFindCert()`.

Declared In `CertificateMgr.h`

Prototype
```
typedef uint32_t CertMgrCertSearchEnum
```

Constants

```
#define apCertMgrSearchCert 1000
```
In this mode, repeated calls to `CertMgrFindCert()` iterate through the certificates in the store, each time returning the certificate that resides at the location indicated by the *index* parameter. Each time `CertMgrFindCert()` is called, the *index* parameter is incremented. Accordingly, you can use this mode to iterate through all of the certificates in the certificate store.

```
#define apCertMgrSearchCertID 1001
```
Causes `CertMgrFindCert()` to look for a certificate with a certID that matches that supplied in the *reference* parameter. The data in *reference* should be a 20-byte certID.

```
#define apCertMgrSearchSubjectRDN
  apCertMgrFieldSubjectRDN
```
Causes `CertMgrFindCert()` to look for a certificate whose SubjectRDN matches the one in the *reference* parameter.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

CertMgrElementListType Struct

Purpose Structure that represents a list of elements. `CertMgrGetField()` fills out this structure.

Declared In `CertificateMgr.h`

Prototype
```
typedef struct {
   uint32_t length;
   uint32_t count;
   CertMgrElementType element[1];
} CertMgrElementListType
```

Fields

`length`
The length of this structure, including all of the `CertMgrElementType` structures needed to contain all of the list's fields.

`count`
The number of fields—that is, the number of `CertMgrElementType` structures—in the list.

`element`
The first element. Subsequent elements follow this one.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

CertMgrElementType Struct

Purpose Structure that represents a single field.

Declared In `CertificateMgr.h`

Prototype

```
typedef struct {
   uint16_t type;
   uint16_t field;
   uint16_t dataType;
   uint16_t length;
   uint32_t offset;
} CertMgrElementType
```

Fields

`type`
The certificate element type. One of the `CertMgrCertElementEnum` values.

`field`
The element field identifier. One of the values listed under "X509Cert Element Fields," "RSA Element Fields," "RDN Element Fields,"or "X509Extensions Element Fields."

`dataType`
The field's data type. One of the values listed under "Data Types" on page 580.

`length`
The length, in bytes, of the field data.

`offset`
The offset, in bytes, to the beginning of the field data.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

CertMgrVerifyResultType Struct

Purpose If a certificate fails to verify during a call to `CertMgrVerifyCert()` or `CertMgrAddCert()`, this structure is filled in and returned to indicate the reason for the verification failure.

Declared In `CertificateMgr.h`

Prototype

```
typedef struct {
   uint32_t failureCode;
   CertMgrCertInfoType cert;
   uint32_t depth;
   uint32_t state;
   DateTimeType verifyTime;
} CertMgrVerifyResultType
```

Fields

`failureCode`
: The reason for the verification failure. See "Certificate Verification Failure Codes" on page 583 for the set of values that can be returned in this field.

`cert`
: The certificate that failed to verify.

`depth`
: How deep the failed certificate is.

`state`
: The verification state.

`verifyTime`
: The date and time against which the certificate was verified.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

Certificate Manager Constants

X509Cert Element Fields

Purpose Fields in an X509Cert element.

Declared In `CertificateMgr.h`

Constants

```
#define apCertMgrElementFieldEntireCert 17
```

```
#define apCertMgrElementFieldExtension 200
#define apCertMgrElementFieldExtensions 23
#define apCertMgrElementFieldInnerDER 1
#define apCertMgrElementFieldIssuerID 15
#define apCertMgrElementFieldIssuerRDN 4
#define apCertMgrElementFieldIssuerUniqueID 21
#define apCertMgrElementFieldNotAfter 7
#define apCertMgrElementFieldNotBefore 6
#define apCertMgrElementFieldPubKeyBER 8
#define apCertMgrElementFieldSerialNumber 3
#define apCertMgrElementFieldSigAlgID 91
#define apCertMgrElementFieldSignature 14
#define apCertMgrElementFieldSigOID 12
#define apCertMgrElementFieldSigParams 13
#define apCertMgrElementFieldSubjectID 16
#define apCertMgrElementFieldSubjectRDN 5
#define apCertMgrElementFieldSubjectUniqueID 22
#define apCertMgrElementFieldVersion 2
#define apCertMgrFieldExtensions
  apCertMgrElementFieldExtensions
#define apCertMgrFieldIssuerID
  apCertMgrElementFieldIssuerID
#define apCertMgrFieldIssuerRDN
  apCertMgrElementFieldIssuerRDN
#define apCertMgrFieldIssuerUniqueID
  apCertMgrElementFieldIssuerUniqueID
#define apCertMgrFieldNotAfter
  apCertMgrElementFieldNotAfter
#define apCertMgrFieldNotBefore
  apCertMgrElementFieldNotBefore
```

```
#define apCertMgrFieldPubKeyBER
  apCertMgrElementFieldPubKeyBER

#define apCertMgrFieldSerialNumber
  apCertMgrElementFieldSerialNumber

#define apCertMgrFieldSignature
  apCertMgrElementFieldSignature

#define apCertMgrFieldSigOID
  apCertMgrElementFieldSigOID

#define apCertMgrFieldSigParams
  apCertMgrElementFieldSigParams

#define apCertMgrFieldSubjectID
  apCertMgrElementFieldSubjectID

#define apCertMgrFieldSubjectRDN
  apCertMgrElementFieldSubjectRDN

#define apCertMgrFieldSubjectUniqueID
  apCertMgrElementFieldSubjectUniqueID

#define apCertMgrFieldVersion
  apCertMgrElementFieldVersion
```

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

RSA Element Fields

Purpose Fields in an RSA element.

Declared In `CertificateMgr.h`

Constants

```
#define apCertMgrElementFieldRSAModulus 16

#define apCertMgrElementFieldRSAPubExpo 17
```

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

RDN Element Fields

Purpose Fields in an RDN element.

Declared In `CertificateMgr.h`

Constants

`#define apCertMgrElementFieldRDNOID 4`
First RDN OID.

`#define apCertMgrElementFieldRDNValue 5`
First RDN value.

Comments To get the second and subsequent OIDs and values, use the `apCertMgrElementFieldRDNOIDN()` and `apCertMgrElementFieldRDNValueN()` macros, respectively.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

X509Extensions Element Fields

Purpose Fields in an X509Extensions element.

Declared In `CertificateMgr.h`

Constants

`#define apCertMgrElementFieldX509ExBytes 2`
First data bytes.

`#define apCertMgrElementFieldX509ExCritical 1`
First critical flag.

`#define apCertMgrElementFieldX509ExOID 0`
First extension OID.

Comments To get the second and subsequent data bytes, critical flags, and extension OIDs, use the `apCertMgrElementFieldX509ExBytesN()`, `apCertMgrElementFieldX509ExCriticalN()`, and `apCertMgrElementFieldX509ExOIDN()` macros, respectively.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

Data Types

Purpose Data type of an element field. These values appear in the CertMgrElementType data structure.

Declared In CertificateMgr.h

Constants

```
#define apCertMgrElementDataTypeASN1BitString 3
#define apCertMgrElementDataTypeASN1BmpString 30
#define apCertMgrElementDataTypeASN1Boolean 1
#define apCertMgrElementDataTypeASN1EmbeddedPDV 11
#define apCertMgrElementDataTypeASN1Enumerated 10
#define apCertMgrElementDataTypeASN1Eoc 0
#define apCertMgrElementDataTypeASN1External 8
#define apCertMgrElementDataTypeASN1GenString 27
#define apCertMgrElementDataTypeASN1GenTime 24
#define apCertMgrElementDataTypeASN1GraphicString
  25
#define apCertMgrElementDataTypeASN1IA5String 22
#define apCertMgrElementDataTypeASN1Integer 2
#define apCertMgrElementDataTypeASN1ISO64String 26
#define apCertMgrElementDataTypeASN1Null 5
#define apCertMgrElementDataTypeASN1NumericString
  18
#define apCertMgrElementDataTypeASN1ObjDesc 7
#define apCertMgrElementDataTypeASN1OctetString 4
#define apCertMgrElementDataTypeASN1OID 6
#define apCertMgrElementDataTypeASN1PrintString 19
#define apCertMgrElementDataTypeASN1Real 9
#define apCertMgrElementDataTypeASN1Sequence 16
#define apCertMgrElementDataTypeASN1Set 17
#define apCertMgrElementDataTypeASN1T61String 20
```

```
#define apCertMgrElementDataTypeASN1UnivString 28
#define apCertMgrElementDataTypeASN1UTCTime 23
#define apCertMgrElementDataTypeASN1UTF8String 12
#define apCertMgrElementDataTypeASN1VideoTexString
  21
```

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

Certificate Formats

Purpose Specifies the certificate format. The `CertMgrCertInfoType` structure's `format` field takes one of these values.

Declared In `CertificateMgr.h`

Constants

`#define apCertMgrFormatX509 1`
The certificate is a DER encoded x509 certificate.

`#define apCertMgrFormatXML 2`
The certificate is formatted as XML.

NOTE: XML-formatted certificates are not currently supported in Palm OS Cobalt.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

Certificate Manager Error Codes

Purpose Error codes returned by the various Certificate Manager functions.

Declared In `CertificateMgr.h`

Constants

`#define certMgrErrBackupInProgress (certErrorClass | 0x0C)`
The certificate vault could not be accessed because it is in the process of being backed up.

`#define certMgrErrBufTooSmall (certErrorClass | 0x07)`
The export buffer is too small. The required size is written into the variable pointed to by the length parameter.

```
#define certMgrErrCertNotFound (certErrorClass |
  0x09)
```
A certificate matching the specified criteria was not found.

```
#define certMgrErrDatabaseFail (certErrorClass |
  0x0B)
```
A Data Manager error occurred.

```
#define certMgrErrFieldNotFound (certErrorClass |
  0x08)
```
The specified field could not be found.

```
#define certMgrErrInvalidEncoding (certErrorClass
  | 0x02)
```
The specified format encoding is invalid.

```
#define certMgrErrInvalidParam (certErrorClass |
  0x04)
```
One of the function parameters is invalid.

```
#define certMgrErrNotExportable (certErrorClass |
  0x0A)
```
The certificate is not exportable. It is probably stored in compressed form.

```
#define certMgrErrNotImplemented (certErrorClass |
  0x01)
```
The requested certificate format is not supported.

```
#define certMgrErrNotRemovable (certErrorClass |
  0x0D)
```
The certificate is not removable.

```
#define certMgrErrOutOfMemory (certErrorClass |
  0x03)
```
There was insufficient memory to complete the operation.

```
#define certMgrErrOutOfResources (certErrorClass |
  0x06)
```
The Certificate Manager ran out of resources.

```
#define certMgrErrServiceNotStarted
  (certErrorClass | 0x05)
```
The Certificate Manager process has not started.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

Certificate Verification Failure Codes

Purpose Indicates why a certificate failed to verify. These values are returned in the `CertMgrVerifyResultType` structure's `failureCode` field as the result of a call to `CertMgrVerifyCert()` or `CertMgrAddCert()`. These are also passed to the SSL Library's Verify callback; see "The Verify Callback" on page 2243.

Declared In `CertificateMgr.h`

Constants

```
#define CertMgrVerifyFail (certErrorClass+0x80)
```

```
#define CertMgrVerifyFailBasicConstraints
  (CertMgrVerifyFail+8)
```

There was a constraint violation.

```
#define CertMgrVerifyFailCriticalExtension
  (CertMgrVerifyFail+9)
```

The critical extension is unknown.

```
#define CertMgrVerifyFailKeyUsage
  (CertMgrVerifyFail+7)
```

```
#define CertMgrVerifyFailNotAfter
  (CertMgrVerifyFail+6)
```

```
#define CertMgrVerifyFailNotBefore
  (CertMgrVerifyFail+5)
```

```
#define CertMgrVerifyFailSelfSigned
  (CertMgrVerifyFail+4)
```

```
#define CertMgrVerifyFailSignature
  (CertMgrVerifyFail+3)
```

The signature is invalid.

```
#define CertMgrVerifyFailUnknown
  (CertMgrVerifyFail+0)
```

```
#define CertMgrVerifyFailUnknownIssuer
  (CertMgrVerifyFail+1)
```

The root cannot be trusted since the issuer is not known.

```
#define CertMgrVerifyFailUnknownSigAlg
  (CertMgrVerifyFail+2)
```

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

Miscellaneous Certificate Manager Constants

Purpose These constants are also declared in `CertificateMgr.h`.

Declared In `CertificateMgr.h`

Constants

```
#define CertMgrServiceName
  "pSysCertificateManager"
```

The name under which the Certificate Manager is registered with the Service Manager.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

Certificate Manager Element Field Macros

apCertMgrElementFieldRDNOIDN Macro

Purpose Macro that evaulates to the field ID for the the second and subsequent OID fields for an RDN element.

Declared In `CertificateMgr.h`

Prototype

```
#define apCertMgrElementFieldRDNOIDN (n)
```

Parameters → *n*

The OID field index. The second OID field's index would be 2.

Returns Evaluates to the RDN element's OID field ID.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

apCertMgrElementFieldRDNValueN Macro

Purpose Macro that evaulates to the field ID for the second and subsequent Value fields for an RDN element..

Declared In `CertificateMgr.h`

Prototype
```
#define apCertMgrElementFieldRDNValueN (n)
```

Parameters → *n*

The Value field index. The second Value field's index would be 2.

Returns Evaluates to the RDN element's Value field ID.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

apCertMgrElementFieldX509ExBytesN Macro

Purpose Macro that evaulates to the field ID for the the second and subsequent Bytes fields for an X509Extended element.

Declared In `CertificateMgr.h`

Prototype
```
#define apCertMgrElementFieldX509ExBytesN (n)
```

Parameters → *n*

The Bytes field index. The second Bytes field's index would be 2.

Returns Evaluates to the X509Extended element's Bytes field ID.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

apCertMgrElementFieldX509ExCriticalN Macro

Purpose Macro that evaulates to the field ID for the the second and subsequent Criticial fields for an X509Extended element.

Declared In `CertificateMgr.h`

Prototype
```
#define apCertMgrElementFieldX509ExCriticalN (n)
```

Parameters → *n*

The Critical field index. The second Critical field's index would be 2.

Returns Evaluates to the X509Extended element's Critical field ID.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

apCertMgrElementFieldX509ExOIDN Macro

Purpose Macro that evaulates to the field ID for the the second and subsequent OID fields for an X509Extended element.

Declared In `CertificateMgr.h`

Prototype

```
#define apCertMgrElementFieldX509ExOIDN (n)
```

Parameters → *n*

The OID field index. The second OID field's index would be 2.

Returns Evaluates to the X509Extended element's OID field ID.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

Certificate Manager Functions and Macros

CertMgrAddCert Function

Purpose Add a certificate to the certificate store.

Declared In `CertificateMgr.h`

Prototype

```
status_t CertMgrAddCert
    (CertMgrCertInfoType *certInfoP,
    Boolean compress,
    CertMgrVerifyResultType *verifyResult)
```

Parameters → *certInfoP*

Pointer to `CertMgrCertInfoType` structure for the certificate to be added.

→ *compress*

If `true`, the certificate is stored in compressed form. This saves space, but note that you cannot export certificates that are compressed.

← *verifyResult*
: Supply a pointer to a CertMgrVerifyResultType structure that will be filled in if the certificate could not be verified.

Returns Returns `errNone` if the certificate was added successfully, or one of the following otherwise:

`certMgrErrInvalidParam`
: *certInfoP* or *verifyResult* is `NULL` .

`certMgrErrServiceNotStarted`
: The Certificate Manager process has not started.

`certMgrErrDatabaseFail`
: A Data Manager error occurred.

`certMgrErrBackupInProgress`
: The certificate vault could not be accessed because it is in the process of being backed up.

Comments This function can be used to add an imported certificate to the certificate store. When a certificate becomes part of the store it is verified and then saved in the Certificate Manager's secure vault. Other applications may then query for it.

If the certificate cannot be added due to a failure in the verification of the certificate, this function fills in the caller-supplied CertMgrVerifyResultType structure. In many cases the caller might choose to override the verification failure and request that the certificate be stored anyway. The only errors that cannot be overridden are signature failure and unknown issuer. To override a verification error, clear the failure code in the `CertMgrVerifyResultType` structure and then call `CertMgrAddCert()` once again.

When the *compress* parameter is set to `true`, some data is thrown away from the certificate to save space. Because of this, the stored certificate is not complete and cannot be exported at a later time.

Example The following code excerpt adds certificates that may be self-signed:

```
while (true) {
   err = CertMgrAddCert(&certInfo, false, &verifyResult);
   if (err) {
      CertMgrReleaseCertInfo(&certInfo);
      goto exit;
```

```
   }

   if (verifyResult.failureCode == 0) {
      break;
   } else {
      if (verifyResult.failureCode ==
         CertMgrVerifyFailSelfSigned) {
         verifyResult.failureCode = 0;
         continue;
      }

      /* Another type of failure */
      break;
   }
}
```

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

See Also `CertMgrImportCert()`, `CertMgrRemoveCert()`, `CertMgrVerifyCert()`

CertMgrExportCert Function

Purpose Exports a certificate from the certificate store to a caller-supplied buffer.

Declared In `CertificateMgr.h`

Prototype

```
status_t CertMgrExportCert
    (CertMgrCertInfoType *certInfoP,
    uint8_t *certData, uint32_t *certDataLen)
```

Parameters

→ *certInfoP*
Reference to the certificate to be exported.

← *certData*
Pointer to a caller-allocated buffer into which the exported certificate data will be written, or `NULL` to determine how large this buffer should be (the needed size, in bytes, is returned via *certDataLen*).

↔ *certDataLen*
When calling this function, `*`*certDataLen* should be set to the size of the *certData* buffer, or 0 to determine how large the *certData* buffer should be. Upon return

`*certDataLen` is set to the size of the exported certificate data.

Returns Returns `errNone` if the operation completed successfully, or one of the following otherwise:

`certMgrErrBufTooSmall`
: The export buffer is too small. The required size is written into `*certDataLen`.

`certMgrErrNotExportable`
: The certificate is not exportable. It is probably stored in compressed form.

`certMgrErrDatabaseFail`
: A Data Manager error occurred.

`certMgrErrBackupInProgress`
: The certificate vault could not be accessed because it is in the process of being backed up.

Comments This function attempts to fill the supplied buffer with the same data that was imported into the Certificate Manager.

NOTE: In Palm OS Cobalt only DER encoded X509 format certificates are supported. Because of this, all exported certificate data will be in this format.

If the certificate was compressed when it was added to the store, it cannot be exported.

If the export buffer is not large enough, an error is returned along with the expected size.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

See Also `CertMgrImportCert()`

CertMgrFindCert Function

Purpose Search the certificate store for a certificate matching the specified criteria.

Declared In `CertificateMgr.h`

Prototype

```
status_t CertMgrFindCert (uint32_t *index,
    CertMgrCertSearchEnum searchFlag,
    uint8_t *reference, uint32_t referenceLen,
    CertMgrCertInfoType *certInfoP)
```

Parameters

↔ *index*
: Set to 0 to start new search. As a certificate matching the specified criteria is found, `*index` is set to the index of the certificate within the certificate store.

→ *searchFlag*
: Value that specifies how the search is to be performed. Supply one of the values listed under `CertMgrCertSearchEnum`.

→ *reference*
: The data being searched for. If *searchFlag* is `apCertMgrSearchCertID`, this should be the 20-byte certID being searched for. If *searchFlag* is `apCertMgrSearchSubjectRDN` search, this should be the SubjectRDN being searched for.

→ *referenceLen*
: The size, in bytes, of the data pointed to by *reference*.

← *certInfoP*
: Pointer to a `CertMgrCertInfoType` structure that is filled in as appropriate to identify the certificate that was found.

Returns Returns `errNone` if the operation completed successfully, or one of the following otherwise:

`certMgrErrInvalidParam`
: One of the function parameters is invalid.

`certMgrErrServiceNotStarted`
: The Certificate Manager process has not started.

`certMgrErrCertNotFound`
: A certificate matching the specified criteria was not found.

`certMgrErrBackupInProgress`
: The certificate vault could not be accessed because it is in the process of being backed up.

Example To find a certificate with the certificate ID in `certificateID`, do the following:

```
err = CertMgrFindCert(0, apCertMgrSearchCertID,
certificateID, 20,
   &certInfo);
```

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

See Also `CertMgrReleaseCertInfo()`

CertMgrGetField Function

Purpose Get the value of a certificate field.

Declared In `CertificateMgr.h`

Prototype

```
status_t CertMgrGetField
   (CertMgrCertInfoType *certInfoP,
   CertMgrCertElementEnum elementType,
   CertMgrCertFieldEnum fieldType,
   CertMgrElementListType *result,
   uint32_t *resultLengthP)
```

Parameters

→ *`certInfoP`*
: Reference to the certificate from which the field is to be retrieved.

→ *`elementType`*
: The certificate element type. This should be one of the values listed under `CertMgrCertElementEnum`.

→ *`fieldType`*
: The type of field to be retrieved. See `CertMgrCertFieldEnum`. Note that the set of values that can be supplied to this parameter varies depending on the value of the *`elementType`* parameter.

← *`result`*
: Pointer to a buffer into which the field data is written, or `NULL` to obtain the size of the needed buffer. The size of the field data (actual or needed) is written into

**resultLengthP*. Note that the contents of the buffer upon return are structured according to CertMgrElementListType.

← *resultLengthP*
When calling this function, **resultLengthP* should be set to the size of the *result* buffer, or 0 to determine how large the *result* buffer should be. Upon return **resultLengthP* is set to the size of the field data.

Returns Returns `errNone` if the operation completed successfully, or one of the following otherwise:

`certMgrErrInvalidParam`
certInfoP is `NULL` or it is not a valid certificate reference, or *resultLengthP* is `NULL`.

`certMgrErrBufTooSmall`
The export buffer is too small. The required size is written into **resultLengthP*.

`certMgrErrFieldNotFound`
The specified field could not be found.

Example The following code excerpt shows you you might use this function.

```
#define CertMgrOrganicationNameOIDLen   5
static uint8_t
CertMgrOrganizationNameOID[CertMgrOrganicationNameOIDLen] =
   {0x06, 0x03, 0x55, 0x04, 0x0a};

/* Getting the issuer RDN from the cert */
CertMgrImplGetField(certInfoP, apCertMgrElementTypeRDN,  0,
   field, &fieldlen);

/* This gets the whole issuer RDN, you must then go through
each field and find the one that you want, an OID field leads
a data field */

/* The following code finds the code for the issuer name and
sets a label to it */
/* Fields in list */
for (index = 0; index < field->count; index++) {

   if (field->element[index].dataType ==
      apCertMgrElementDataTypeASN1OID) {
      if (field->element[index].length ==
         CertMgrOrganicationNameOIDLen) {
```

```
            if (MemCmp(CertMgrOrganizationNameOID,
               ((uint8_t *)field)+field->element[index].offset,
               CertMgrOrganizationNameOIDLen) == 0) {

               uint16_t buflen = field->element[index +
                  1].length;
               uint8_t *buffer = ((uint8_t *)field) +
                  field->element[index + 1].offset;
               uint16_t count  = 0;

               Char label[40];
               uint16_t pre = 0;

               MemSet(label, 40, 0);

               if (buflen < 40) {
                  pre = (40 - buflen) / 2;

                  MemSet(label, pre, ' ');
               }

               MemMove(label+pre,buffer,buflen>39 ? 39:buflen);

               FrmCopyLabel(frmP, selfsignedaddCertnameLabel,
                  label);
               break;
            }
         }
      }
}
```

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

CertMgrImportCert Function

Purpose Imports a certificate from a buffer into the certificate store.

Declared In `CertificateMgr.h`

Prototype

```
status_t CertMgrImportCert (uint8_t *certData,
   uint32_t certDataLen,
   CertMgrCertInfoType *certInfoP)
```

Parameters

→ *`certData`*
Pointer to a buffer containing the certificate data being imported.

→ *`certDataLen`*
The size, in bytes, of the data in *`certData`*.

↔ *`certInfoP`*
When calling this function, you can optionally specify the format of the certificate data by setting the `format` field of this structure. Upon return, this structure's fields are filled in appropriately to identify the certificate that was imported.

NOTE: In Palm OS Cobalt only DER encoded X509 format certificates are supported.

Returns Returns `errNone` if the operation completed successfully, or one of the following otherwise:

`certMgrErrInvalidEncoding`
The specified format encoding is invalid.

`certMgrErrOutOfMemory`
There was insufficient memory to complete the operation.

`certMgrErrInvalidParam`
One of the function parameters is invalid.

`certMgrErrServiceNotStarted`
The Certificate Manager process has not started.

`certMgrErrBackupInProgress`
The certificate vault could not be accessed because it is in the process of being backed up.

`certMgrErrNotImplemented`
The requested certificate format is not supported.

Example A likely scenario would be an application that imports a certificate that was stored in the its PRC. The following code excerpt shows how to do this:

```
/* Get certificate from PRC */
err = SignGetCertificateByIndex(dbP, certIndex, &certBlock,
   &certDataLength, certData);

/* Load certificate onto cert mgr */
err = CertMgrImportCert(certData, certDataLength, &certInfo);
```

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

See Also `CertMgrExportCert()`

CertMgrReleaseCertInfo Function

Purpose Release resources that allocated by the Certificate Manager during a successful call to `CertMgrFindCert()` or `CertMgrImportCert()`.

Declared In `CertificateMgr.h`

Prototype

```
status_t CertMgrReleaseCertInfo
    (CertMgrCertInfoType *certInfoP)
```

Parameters → *`certInfoP`*
Reference to the certificate for which resources are to be released..

Returns Returns `errNone` if the operation completed successfully, or `certMgrErrInvalidParam` if *`certInfoP`* is invalid.

Comments Failure to call this function after a successful call to `CertMgrFindCert()` or `CertMgrImportCert()` will result in a memory leak.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

CertMgrRemoveCert Function

Purpose Remove a certificate from the certificate store.

Declared In `CertificateMgr.h`

Prototype

```
status_t CertMgrRemoveCert
    (CertMgrCertInfoType *certInfoP)
```

Parameters ↔ *`certInfoP`*
Pointer to `CertMgrCertInfoType` structure for the certificate to be added.

Returns Returns `errNone` if the operation completed successfully, or one of the following otherwise:

`certMgrErrInvalidParam`
`certInfoP` is invalid.

`certMgrErrBackupInProgress`
: The certificate vault could not be accessed because it is in the process of being backed up.

`certMgrErrNotRemovable`
: The certificate is not removable.

Comments This function does not invalidate any other certificates already in the store. However, it may cause further verifications of new certificates to fail (for instance, if a root certificate is removed).

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

See Also `CertMgrAddCert()`

CertMgrVerifyCert Function

Purpose Authenticate the validity of a certificate.

Declared In `CertificateMgr.h`

Prototype

```
status_t CertMgrVerifyCert
    (CertMgrCertInfoType *certInfoP,
    CertMgrCertChainType *certChainP,
    CertMgrVerifyResultType *verifyResult)
```

Parameters

→ *certInfoP*
: Pointer to `CertMgrCertInfoType` structure for the certificate to be validated.

→ *certChainP*
: A chain of certificates that make up the authentication tree for this certificate.

← *verifyResult*
: Supply a pointer to a `CertMgrVerifyResultType` structure that will be filled in if the certificate could not be verified.

Returns Returns `errNone` if the operation completed successfully, or one of the following otherwise:

`certMgrErrInvalidParam`
: *certInfoP* or *verifyResult* is `NULL`.

`certMgrErrOutOfMemory`
: There was insufficient memory to complete the operation.

`certMgrErrOutOfResources`
: The Certificate Manager ran out of resources.

`certMgrErrServiceNotStarted`
: The Certificate Manager process has not started.

This function authenticates the validity of the certificate. Many different error conditions may occur, and they are returned through the *`verifyResult`* parameter.

Specify a certificate chain when the chain of certificates that authenticate the specified certificate is not contained in the Certificate Manager's certificate store. (For instance, when all or some of the certificatess in the chain have just been imported but not yet added.) When necessary, the Certificate Manager will also authenticate a certificate in the chain. The chain need not be in any order, and the certificates in the chain need not all be part of the chain.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

See Also `CertMgrAddCert()`

CertMgrVerifyFailure Macro

Purpose Determine if a given error code is a Certificate Manager verification error.

Declared In `CertificateMgr.h`

Prototype

```
#define CertMgrVerifyFailure (err)
```

Parameters

→ *`err`*
: The error code being checked.

Returns Evaluates to `true` if the supplied error code is within the range of verification failure errors, `false` otherwise. See "Certificate Verification Failure Codes" on page 583 for those error codes that are classified as verification errors.

Compatibility Implemented only if Palm OS Cobalt Feature Set is present.

29

Character Attributes

This chapter provides reference material for character attributes functions defined in `CharAttr.h`.

Character Attribute Functions

ChrHorizEllipsis Macro

Purpose Returns the appropriate character code for the horizontal ellipsis.

Declared In `Chars.h`

Prototype

```
#define ChrHorizEllipsis (chP)
```

Parameters ← *chP*

Pointer to a variable in which to return the horizontal ellipsis character code.

Returns Returns nothing. Upon return, the variable pointed to by `chP` contains the horizontal ellipsis character.

Comments Version 3.1 of the Palm OS® uses different character codes for the horizontal ellipsis character and the numeric space character than earlier versions did. Use this macro to return the appropriate code for horizontal ellipsis regardless of which version of Palm OS your application is run on.

ChrIsHardKey Macro

Purpose Returns true if the character is one of the hard keys on the device.

Declared In `Chars.h`

Prototype

```
#define ChrIsHardKey (ch)
```

Parameters → *ch*

The character from the `keyDownEvent`.

Returns `true` if the character is one of the four built-in hard keys on the device, `false` otherwise.

Compatibility This macro is obsolete and replaced by `TxtCharIsHardKey()` if the International Feature Set is present.

ChrNumericSpace Macro

Purpose Returns the appropriate character code for the numeric space.

Declared In `Chars.h`

Prototype

```
#define ChrNumericSpace (chP)
```

Parameters

← *chP*
Pointer to a variable in which to return the numeric space character code.

Returns Returns nothing. Upon return, the variable pointed to by `chP` contains the numeric space character.

Comments Version 3.1 of the Palm OS uses different character codes for the horizontal ellipsis character and the numeric space character than earlier versions did. Use this macro to return the appropriate code for numeric space regardless of which version of Palm OS your application is run on.

GetCharAttr Function

Purpose Return a pointer to the character attribute array. This array is used by the character classification and character conversion macros (such as `isalpha`).

Declared In `CharAttr.h`

Prototype

```
UInt16 *GetCharAttr (void)
```

Parameters None

Returns A pointer to the attributes array. This is an array of 256 `UInt16` values, one for each possible character code. See `CharAttr.h` for an explanation of the attributes.

Compatibility This function is *not* implemented if International Feature Set is present.

NOTE: This function is provided for backwards compatibility only. Use Text Manager functions instead on systems that support the text manager.

If 5.0 New Feature Set is present, this function is implemented in PACE. However, it only supports the Latin table, regardless of the localization.

See Also `TxtCharAttr()`, `TxtCharXAttr()`

GetCharCaselessValue Function

Purpose Return a pointer to an array that maps all characters to an assigned caseless and accentless value. Use this function for finding text.

Declared In `CharAttr.h`

Prototype

```
UInt8 *GetCharCaselessValue (void)
```

Parameters None.

Returns Returns a pointer to the sort array, which is an array of 256 bytes.

Comments The `GetCharCaselessValue` conversion table converts each character into a numeric value that is caseless and sorted according to Microsoft Windows sorting rules:

- Punctuation characters have the lowest values,
- followed by numbers,
- followed by alpha characters.

  All forms of each alpha character have equivalent values, so that e = E = e-grave = e-circumflex, etc.

This conversion table is used by all the Palm OS sorting and comparison routines to yield caseless searches and caseless sorts in the almost same order as Windows-based programs, except that Palm OS routines produce the same sorting for all locales.

Compatibility This function is *not* implemented if International Feature Set is present.

NOTE: This function is provided for backwards compatibility only. Use Text Manager functions instead on systems that support the text manager.

If 5.0 New Feature Set is present, this function is implemented in PACE. However, it only supports the Latin table, regardless of the localization.

GetCharSortValue Function

Purpose Return a pointer to an array that maps all characters to an assigned sorting value. Use this function for ordering (sorting) text.

Declared In `CharAttr.h`

Prototype

```
UInt8 *GetCharSortValue (void)
```

Parameters None.

Returns Returns a pointer to the attributes array. This is an array of 256 `UInt8` values, one for each possible character code.

Compatibility This function is *not* implemented if International Feature Set is present.

NOTE: This function is provided for backwards compatibility only. Use Text Manager functions instead on systems that support the text manager.

If 5.0 New Feature Set is present, this function is implemented in PACE. However, it only supports the Latin table, regardless of the localization.

30

Data and Resource Manager

This chapter describes the data manager and the resource manager API declared in the header file `DataMgr.h`. It discusses the following topics:

- Data Manager Data Structures
- Data Manager Constants
- Data Manager Functions
- Application-Defined Functions

For more information on the data and resource managers, see the chapter "Files and Databases" in the *Palm OS Programmer's Companion*, vol. I.

Data Manager Data Structures

DmOpenRef Typedef

Purpose The `DmOpenRef` type defines a pointer to an open database. The database pointer is created and returned by `DmOpenDatabase()`. It is used in any function that requires access to an open database.

Prototype

```
typedef void *DmOpenRef;
```

DmResID Typedef

Purpose The `DmResID` type defines a resource identifier. You assign each resource an ID at creation time. Note that resource IDs greater than or equal to 10000 are reserved for system use.

Prototype

```
typedef UInt16 DmResID;
```

DmResType Typedef

Purpose The `DmResType` type defines the type of a resource. The resource type is a four-character code such as `'Tbmp'` for bitmap resources.

Prototype

```
typedef UInt32 DmResType;
```

SortRecordInfoType Struct

Purpose The `SortRecordInfoType` structure specifies information that can be used to sort a record. The database sorting functions (`DmInsertionSort()` and `DmQuickSort()`) pass this structure to your comparison callback function (of type `DmComparF()`), where you can use the information therein to help when comparing two records. To create this structure, you can call `DmRecordInfo()`, which returns these values for a given record.

Prototype

```
typedef struct {
  UInt8 attributes;
  UInt8 uniqueID[3];
} SortRecordInfoType;
typedef SortRecordInfoType *SortRecordInfoPtr;
```

Fields

`attributes`
: The record's attributes. See "Record Attribute Constants."

`uniqueID`
: The unique identifier for the record.

Data Manager Constants

Category Constants

The following constants are used to specify information about categories:

Constant	Value	Description
`dmAllCategories`	0xFF	A mask used to represent all categories.
`dmCategoryLength`	16	The length of a category name. Currently, this is 16 bytes, which includes the null terminator.
`dmRecAttrCategoryMask`	0x0F	A mask used to retrieve the category information from the record's attributes field.
`dmRecNumCategories`	16	The number of categories allowed. Currently, this is 16, which includes the "Unfiled" category.
`dmUnfiledCategory`	0	A mask used to indicate the Unfiled category.

Record Attribute Constants

The following constants specify a database record's attributes.

Constant	Value	Description
`dmMaxRecordIndex`	0xFFFF	Indicates the highest record index allowed.
`dmAllRecAttrs`	0xF0	A mask used to specify all record attributes.
`dmRecAttrBusy`	0x20	Busy (the application has locked access to this record). A call to `DmGetRecord()` fails on a record that has this bit set, otherwise it sets this bit. Call `DmReleaseRecord()` to release the record and clear this bit. The `DmSetRecordInfo()` function cannot be used to alter the state of `dmRecAttrBusy`.
`dmRecAttrDelete`	0x80	Deleted
`dmRecAttrDirty`	0x40	Dirty (has been modified since last sync)

Constant	Value	Description
`dmRecAttrSecret`	0x10	Private
`dmSysOnlyRecAttrs`	0x20	A mask used to specify record attributes that only the system can change. (In other words, the busy attribute.)

Database Attribute Constants

The following constants define a database's attributes:

Constant	Description
`dmAllHdrAttrs`	A mask used to specify all header attributes.
`dmHdrAttrAppInfoDirty`	The application info block is dirty (has been modified since the last sync).
`dmHdrAttrBackup`	The database should be backed up to the desktop computer if no application-specific conduit is available.
`dmHdrAttrBundle`	The database is bundled with its application during a beam. That is, if the user chooses to beam the application from the Launcher, the Launcher beams this database along with the application's resource database and overlay database. This attribute applies to Palm OS® 4.0 and higher. Note that overlay databases are automatically beamed with the application database on Palm OS 4.0 and higher. You do not need to set this bit in overlay databases.
`dmHdrAttrCopyPrevention`	Prevents the database from being copied by methods such as IR beaming.

Constant	Description
`dmHdrAttrHidden`	This database should be hidden from view. For example, this attribute is set to hide some applications in the launcher's main view. You can set it on record databases to have the launcher disregard the database's records when showing a count of records. This attribute applies to Palm OS version 3.2 and higher.
`dmHdrAttrLaunchableData`	This database is a data database but it can be "launched" from the launcher. For example, this attribute is set in Palm Query Applications (PQAs) launched by the Web Clipping Application Viewer application.
`dmHdrAttrOpen`	The database is open.
`dmHdrAttrOKToInstallNewer`	The backup conduit can install a newer version of this database with a different name if the current database is open. This mechanism is used to update the Graffiti® and Graffiti 2 Shortcuts databases, for example.
`dmHdrAttrReadOnly`	The database is a read-only database.
`dmHdrAttrRecyclable`	The database is recyclable. Recyclable databases are deleted when they are closed or upon a system reset. This attribute applies to Palm OS 4.0 and higher.
`dmHdrAttrResDB`	The database is a resource database.
`dmHdrAttrResetAfterInstall`	The device must be reset after this database is installed. That is, the HotSync® application forces a reset after installing this database.

Constant	Description
`dmHdrAttrStream`	The database is a file stream.
`dmSysOnlyHdrAttrs`	A mask specifying the attributes that only the system can change (open and resource database).

Error Codes

The following constants define error codes that are returned by the data manager and resource manager functions. Several functions return a failure value such as `NULL` or 0 instead of an error code. In many cases, you can call `DmGetLastErr()` upon receiving this value and receive a more descriptive error code.

Also, note that on releases prior to Palm OS release 3.5, many data manager functions display a fatal error message using the `ErrFatalDisplayIf()` macro if certain error conditions are true. Because the Palm OS ROMs are usually shipped with error checking set to partial, you receive the fatal error message. If a ROM is built with error checking set to none, the function returns one of the error codes listed here. (Note that Palm™ has never released a ROM with error checking set to none and has no plans to do so.)

Constant	Description
`dmErrAlreadyExists`	Another database with the same name already exists in RAM store.
`dmErrAlreadyOpenForWrites`	The database is already open with write access.
`dmErrCantFind`	The specified resource can't be found.
`dmErrCantOpen`	The database cannot be opened.
`dmErrCorruptDatabase`	The database is corrupted.

Constant	Description
dmErrDatabaseOpen	The function cannot be performed on an open database, and the database is open.
dmErrDatabaseNotProtected	DmDatabaseProtect() failed to protect the specified database.
dmErrDatabaseProtected	The database is marked as protected.
dmErrIndexOutOfRange	The specified index is out of range.
dmErrInvalidDatabaseName	The name you've specified for the database is invalid.
dmErrInvalidParam	The function received an invalid parameter.
dmErrMemError	A memory error occurred.
dmErrNoOpenDatabase	The function is to search all open databases, but there are none.
dmErrNotRecordDB	You've attempted to perform a record function on a resource database.
dmErrNotResourceDB	You've attempted to perform a resource manager function on a record database.
dmErrNotValidRecord	The record handle is invalid.
dmErrOpenedByAnotherTask	You've attempted to open a database that another task already has open.

Constant	Description
`dmErrReadOnly`	You've attempted to write to or modify a database that is in read-only mode.
`dmErrRecordArchived`	The function requires that the record not be archived, but it is.
`dmErrRecordBusy`	The function requires that the record not be busy, but it is.
`dmErrRecordDeleted`	The record has been deleted.
`dmErrRecordInWrongCard`	You've attempted to attach a record to a database when the record and database reside on different memory cards.
`dmErrResourceNotFound`	The resource can't be found.
`dmErrROMBased`	You've attempted to delete or modify a ROM-based database.
`dmErrSeekFailed`	The operation of seeking the next record in the category failed.
`dmErrUniqueIDNotFound`	A record with the specified unique ID can't be found.
`dmErrWriteOutOfBounds`	A write operation exceeded the bounds of the record.
`memErrCardNotPresent`	The specified card can't be found.

Constant	Description
`memErrChunkLocked`	The associated memory chunk is locked.
`memErrInvalidParam`	A memory error occurred.
`memErrNotEnoughSpace`	
`memErrInvalidStoreHeader`	The specified card has no storage RAM.
`memErrRAMOnlyCard`	
`omErrBaseRequiresOverlay`	An attempt was made to open a stripped resource database, but no associated overlay could be found.
`omErrUnknownLocale`	An attempt was made to open a resource database with overlays using an unknown locale.

Open Mode Constants

The following constants define the mode in which a database can be opened. You pass one or more of these as a parameter to `DmOpenDatabase()`, `DmOpenDatabaseByTypeCreator()`, or `DmOpenDBNoOverlay()`:

Constant	Description
`dmModeReadWrite`	Read-write access.
`dmModeReadOnly`	Read-only access.
`dmModeWrite`	Write-only access.
`dmModeLeaveOpen`	Leave database open even after application quits.

Constant	Description
`dmModeExclusive`	Don't let anyone else open this database.
`dmModeShowSecret`	Show records marked private.

Miscellaneous Constants

The following additional constants are used in conjunction with the Data Manager.

Constant	Value	Description
`dmDBNameLength`	32	Maximum length of a database name, including the null terminator. Database names must use only 7-bit ASCII characters (0x20 through 0x7E).

Data Manager Functions

DmArchiveRecord Function

Purpose Mark a record as archived by leaving the record's chunk intact and setting the delete bit for the next sync.

Declared In `DataMgr.h`

Prototype

```
Err DmArchiveRecord (DmOpenRef dbP, UInt16 index)
```

Parameters

→ *dbP*
: `DmOpenRef` to open database.

→ *index*
: Which record to archive.

Returns Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrReadOnly`

- dmErrIndexOutOfRange
- dmErrRecordArchived
- dmErrRecordDeleted
- memErrInvalidParam

Some releases may display a fatal error message instead of returning the error code.

Comments When a record is archived, the deleted bit is set but the chunk is not freed and the local ID is preserved. This way, the next time the user synchronizes with the desktop system, the desktop can save the record data on the PC before it permanently removes the record entry and data from the Palm Powered™ device.

Based on the assumption that a call to `DmArchiveRecord` indicates that you are finished with the record and aren't going to refer to it again, this function sets the chunk's lock count to zero.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also DmRemoveRecord(), DmDetachRecord(), DmNewRecord(), DmDeleteRecord()

DmAttachRecord Function

Purpose Attach an existing chunk ID handle to a database as a record.

Declared In `DataMgr.h`

Prototype

```
Err DmAttachRecord (DmOpenRef dbP, UInt16 *atP,
    MemHandle newH, MemHandle *oldHP)
```

Parameters

→ *dbP*
: `DmOpenRef` to open database.

↔ *atP*
: Pointer to the index where the new record should be placed. Specify the value `dmMaxRecordIndex` to add the record to the end of the database.

→ *newH*
Handle of the new record.

↔ *oldHP*
If non-NULL upon entry, indicates that the record at *atP should be replaced. Upon return, contains the handle to the replaced record.

Returns Returns errNone if no error, or one of the following if an error occurs:

- dmErrMemError
- memErrChunkLocked
- memErrInvalidParam
- memErrNotEnoughSpace
- dmErrReadOnly
- dmErrNotRecordDB
- dmErrRecordInWrongCard
- dmErrIndexOutOfRange

Some releases may display a fatal error message instead of returning some of these error codes.

Comments Given the handle of an existing chunk, this routine makes that chunk a new record in a database and sets the dirty bit. The parameter atP points to an index variable. If oldHP is NULL, the new record is inserted at index *atP and all record indices that follow are shifted down. If *atP is greater than the number of records currently in the database, the new record is appended to the end and its index is returned in *atP. If oldHP is not NULL, the new record replaces an existing record at index *atP and the handle of the old record is returned in *oldHP so that the application can free it or attach it to another database.

This function is useful for cutting and pasting between databases.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database.

For more information see "Accessing the PIM Application Databases" on page 2451.

See Also DmDetachRecord(), DmNewRecord(), DmNewHandle(), DmFindSortPosition()

DmAttachResource Function

Purpose Attach an existing chunk ID to a resource database as a new resource.

Declared In DataMgr.h

Prototype

```
Err DmAttachResource (DmOpenRef dbP,
    MemHandle newH, DmResType resType,
    DmResID resID)
```

Parameters

→ *dbP*
DmOpenRef to open database.

→ *newH*
Handle of new resource's data.

→ *resType*
Type of the new resource.

→ *resID*
ID of the new resource.

Returns Returns errNone if no error, or one of the following if an error occurs:

- dmErrMemError
- memErrChunkLocked
- memErrInvalidParam
- memErrNotEnoughSpace
- dmErrReadOnly
- dmErrRecordInWrongCard

Some releases may display a fatal error message instead of returning some of these error codes. All releases may display a fatal error message if the database is not a resource database.

Comments Given the handle of an existing chunk with resource data in it, this routine makes that chunk a new resource in a resource database. The new resource will have the given type and ID.

See Also DmDetachResource(), DmRemoveResource(), DmNewHandle(), DmNewResource()

DmCloseDatabase Function

Purpose Close a database.

Declared In `DataMgr.h`

Prototype

```
Err DmCloseDatabase (DmOpenRef dbP)
```

Parameters → *dbP*
Database access pointer.

Returns Returns `errNone` if no error, or `dmErrInvalidParam` if an error occurs. Some releases may display a fatal error message instead of returning the error code.

Comments This routine doesn't unlock any records that were left locked. Records and resources should not be left locked. If a record/resource is left locked, you should not use its reference because the record can disappear during a HotSync operation or if the database is deleted by the user. To prevent the database from being deleted, you can use `DmDatabaseProtect()` before closing.

If there is an overlay associated with the database passed in, `DmCloseDatabase` closes the overlay as well.

If the database has the recyclable bit set (`dmHdrAttrRecyclable`), `DmCloseDatabase` calls `DmDeleteDatabase()` to delete it.

Compatibility Starting with Palm OS 2.0, `DmCloseDatabase` updates the database's modification date.

- On Palm OS 2.0, the modification date is updated if the database was opened with write access.
- On Palm OS 3.0 and higher, the modification date is updated only if a change has been made and the database was opened with write access. Changes that trigger an update include adding, deleting, archiving, rearranging, or resizing records, setting a record's dirty bit in `DmReleaseRecord()`,

rearranging or deleting categories, or updating the database header fields using DmSetDatabaseInfo().

Under Palm OS 1.0, the modification date was never updated.

If you need to ensure that the modification date is updated the same way regardless of the operating system version, use DmSetDatabaseInfo() to set the modification date explicitly.

If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also DmOpenDatabase(), DmDeleteDatabase(), DmOpenDatabaseByTypeCreator()

DmCreateDatabase Function

Purpose Create a new database on the specified card with the given name, creator, and type.

Declared In DataMgr.h

Prototype

```
Err DmCreateDatabase (UInt16 cardNo,
   const Char *nameP, UInt32 creator,
   UInt32 type,
   Boolean resDB)
```

Parameters

→ *cardNo*
The card number to create the database on.

→ *nameP*
Name of new database, up to 32 ASCII bytes long, including the null terminator (as specified by dmDBNameLength). Database names must use only 7-bit ASCII characters (0x20 through 0x7E).

→ *creator*
Creator of the database.

→ *type*
Type of the database.

→ *resDB*
If true, create a resource database.

Returns Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrInvalidDatabaseName`
- `dmErrAlreadyExists`
- `memErrCardNotPresent`
- `dmErrMemError`
- `memErrChunkLocked`
- `memErrInvalidParam`
- `memErrInvalidStoreHeader`
- `memErrNotEnoughSpace`
- `memErrRAMOnlyCard`

May display a fatal error message if the master database list cannot be found.

Comments Call this routine to create a new database on a specific card. If another database with the same name already exists in RAM store, this routine returns a `dmErrAlreadyExists` error code. Once created, the database ID can be retrieved by calling `DmFindDatabase()`. The database can be opened using the database ID. To create a resource database instead of a record-based database, set the `resDB` Boolean to `true`.

After you create a database, it's recommended that you call `DmSetDatabaseInfo()` to set the version number. Databases default to version 0 if the version isn't explicitly set.

See Also `DmCreateDatabaseFromImage()`, `DmOpenDatabase()`, `DmDeleteDatabase()`

DmCreateDatabaseFromImage Function

Purpose Create an entire database from a single resource that contains an image of the database.

Declared In `DataMgr.h`

Prototype

```
Err DmCreateDatabaseFromImage (MemPtr bufferP)
```

Parameters

→ *bufferP*
Pointer to locked resource containing database image.

Returns Returns `errNone` if no error.

Comments An image is the same as a desktop file representation of a prc or pdb file.

This function is intended for applications in the ROM to install default databases after a hard reset. RAM-based applications that want to install a default database should install a pdb file separately to save storage heap space.

See Also `DmCreateDatabase()`, `DmOpenDatabase()`

DmDatabaseInfo Function

Purpose Retrieve information about a database.

Declared In `DataMgr.h`

Prototype

```
Err DmDatabaseInfo (UInt16 cardNo, LocalID dbID,
    Char *nameP, UInt16 *attributesP,
    UInt16 *versionP, UInt32 *crDateP,
    UInt32 *modDateP, UInt32 *bckUpDateP,
    UInt32 *modNumP, LocalID *appInfoIDP,
    LocalID *sortInfoIDP, UInt32 *typeP,
    UInt32 *creatorP)
```

Parameters

→ *cardNo*
Number of the card the database resides on.

→ *dbID*
Database ID of the database.

← *nameP*
The database's name. Pass a pointer to 32-byte character array for this parameter, or `NULL` if you don't care about the name.

← *attributesP*
The database's attribute flags. The section "Database Attribute Constants" lists constants you can use to query the values returned in this parameter. Pass `NULL` for this parameter if you don't want to retrieve it.

← *versionP*
The application-specific version number. The default version number is 0. Pass `NULL` for this parameter if you don't want to retrieve it.

← *crDateP*
The date the database was created, expressed as the number of seconds since the first instant of Jan. 1, 1904. Pass `NULL` for this parameter if you don't want to retrieve it.

← *modDateP*
The date the database was last modified, expressed as the number of seconds since the first instant of Jan. 1, 1904. Pass `NULL` for this parameter if you don't want to retrieve it.

← *bckUpDateP*
The date the database was backed up, expressed as the number of seconds since the first instant of Jan. 1, 1904. Pass `NULL` for this parameter if you don't want to retrieve it.

← *modNumP*
The modification number, which is incremented every time a record in the database is added, modified, or deleted. Pass `NULL` for this parameter if you don't want to retrieve it.

← *appInfoIDP*
The local ID of the application info block, or `NULL`. The application info block is an optional field that the database may use to store application-specific information about the database. Pass `NULL` for this parameter if you don't want to retrieve it.

← *sortInfoIDP*
The local ID of the database's sort table. This is an optional field in the database header. Pass `NULL` for this parameter if you don't want to retrieve it.

← *typeP*
The database's type, specified when it is created. Pass `NULL` for this parameter if you don't want to retrieve it.

← *creatorP*
The database's creator, specified when it is created. Pass `NULL` for this parameter if you don't want to retrieve it.

Returns Returns `errNone` if no error, or `dmErrInvalidParam` if an error occurs.

Compatibility Updating of the modification date differs based on which version of the OS your application is running on.

- Under Palm OS 1.0, the modification date is never updated.
- Under Palm OS 2.0, the modification date is updated every time a database opened with write access is closed.
- Beginning with Palm OS 3.0, the modification date is updated only if a change has been made to the database opened with write access. (The update still occurs upon closing the database.) Changes that trigger an update include adding, deleting, archiving, rearranging, or resizing records, setting a record's dirty bit in `DmReleaseRecord()`, rearranging or deleting categories, or updating the database header fields using `DmSetDatabaseInfo()`.

If you need to ensure that the modification date is updated the same way regardless of the operating system version, use `DmSetDatabaseInfo()` to set the modification date explicitly.

See Also `DmSetDatabaseInfo()`, `DmDatabaseSize()`, `DmOpenDatabaseInfo()`, `DmFindDatabase()`, `DmGetNextDatabaseByTypeCreator()`, `TimSecondsToDateTime()`

DmDatabaseProtect Function

Purpose Increment or decrement the database's protection count.

Declared In `DataMgr.h`

Prototype

```
Err DmDatabaseProtect (UInt16 cardNo,
   LocalID dbID, Boolean protect)
```

Parameters → *cardNo*
Card number of database to protect/unprotect.

→ *dbID*
Local ID of database to protect/unprotect.

→ *protect*
If `true`, `protect` count will be incremented. If `false`, `protect` count will be decremented.

Returns Returns `errNone` if no error, or one of the following if an error occurs:

- `memErrCardNotPresent`
- `dmErrROMBased`
- `dmErrCantFind`
- `memErrNotEnoughSpace`
- `dmErrDatabaseNotProtected`

Comments This routine can be used to prevent a database from being deleted (by passing `true` for the `protect` parameter). It increments the protect count if `protect` is `true` and decrements it if `protect` is `false`. All `true` calls should be balanced by `false` calls before the application terminates.

Use this function if you want to keep a particular record or resource in a database locked down but don't want to keep the database open. This information is kept in the dynamic heap, so all databases are "unprotected" at system reset.

If the database is a resource database that has an overlay associated with it for the current locale, the overlay is also protected or unprotected by this call.

Compatibility Implemented only if 2.0 New Feature Set is present. Overlay support is only available if 3.5 New Feature Set is present.

DmDatabaseSize Function

Purpose Retrieve size information on a database.

Declared In `DataMgr.h`

Prototype

```
Err DmDatabaseSize (UInt16 cardNo, LocalID dbID,
   UInt32 *numRecordsP, UInt32 *totalBytesP,
   UInt32 *dataBytesP)
```

Parameters → *cardNo*
Card number the database resides on.

→ *dbID*
Database ID of the database.

← *numRecordsP*
The total number of records in the database. Pass `NULL` for this parameter if you don't want to retrieve it.

← *totalBytesP*
The total number of bytes used by the database including the overhead. Pass `NULL` for this parameter if you don't want to retrieve it.

← *dataBytesP*
The total number of bytes used to store just each record's data, not including overhead. Pass `NULL` for this parameter if you don't want to retrieve it.

Returns Returns `errNone` if no error, or `dmErrMemError` if an error occurs.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmDatabaseInfo()`, `DmOpenDatabaseInfo()`, `DmFindDatabase()`, `DmGetNextDatabaseByTypeCreator()`

DmDeleteCategory Function

Purpose Delete all records in a category. The category name is not changed.

Declared In `DataMgr.h`

Prototype

```
Err DmDeleteCategory (DmOpenRef dbR,
   UInt16 categoryNum)
```

Parameters

→ *dbR*
Database access pointer.

→ *categoryNum*
Category of records to delete. Category masks such as `dmAllCategories` are invalid.

Returns Returns `errNone` if no error, or one of the following if an error occurs:

- dmErrReadOnly
- memErrInvalidParam

Some releases may display a fatal error message instead of returning the error code.

Comments

This function deletes all records in a category, but does not delete the category itself (note that it deletes the record data and header info, and doesn't just set the deleted bit). For each record in the category, `DmDeleteCategory` marks the `delete` bit in the database header for the record and disposes of the record's data chunk. The record entry in the database header remains, but its `localChunkID` is set to `NULL`.

If the category contains no records, this function does nothing and returns `errNone` to indicate success. The `categoryNum` parameter is assumed to represent a single category. If you pass a category mask to specify more than one category, this function interprets that value as a single category, finds no records to delete in that category, and returns `errNone`.

You can use the `DmRecordInfo()` call to obtain a category index from a given record. For example:

```
DmOpenRef myDB;
UInt16 record, attr, category, total;

DmRecordInfo(myDB, record, &attr, NULL, NULL);
category = attr & dmRecAttrCategoryMask;
err = DmDeleteCategory(myDB, category);
```

Compatibility

Implemented only if 2.0 New Feature Set is present.

If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, `sysErrNotAllowed` is returned. For more information see "Accessing the PIM Application Databases" on page 2451.

DmDeleteDatabase Function

Purpose Delete a database and all its records.

Declared In `DataMgr.h`

Prototype

```
Err DmDeleteDatabase (UInt16 cardNo,
   LocalID dbID)
```

Parameters

→ *cardNo*
Card number the database resides on.

→ *dbID*
Database ID.

Returns Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrCantFind`
- `dmErrCantOpen`
- `memErrChunkLocked`
- `dmErrDatabaseOpen`
- `dmErrDatabaseProtected`
- `dmErrROMBased`
- `memErrInvalidParam`
- `memErrNotEnoughSpace`

Comments Call this routine to delete a database. This routine deletes the database, the application info block, the sort info block, and any other overhead information that is associated with this database. After deleting the database, this function enqueues a deferred `sysNotifyDBDeletedEvent` notification, which will be broadcast at the top of the event loop.

If the database has an overlay associated with it, this function does *not* delete the overlay. You can delete the overlay with a separate call to `DmDeleteDatabase`.

This routine accepts a database ID as a parameter. To determine the database ID, call either `DmFindDatabase()` or `DmGetDatabase()` with a database index.

Compatibility The `sysNotifyDBDeletedEvent` notification is only broadcast if the 4.0 New Feature Set is present.

See Also `DmDeleteRecord()`, `DmRemoveRecord()`, `DmRemoveResource()`, `DmCreateDatabase()`, `DmGetNextDatabaseByTypeCreator()`, `DmFindDatabase()`

DmDeleteRecord Function

Purpose Delete a record's chunk from a database but leave the record entry in the header and set the `delete` bit for the next sync.

Declared In `DataMgr.h`

Prototype

```
Err DmDeleteRecord (DmOpenRef dbP, UInt16 index)
```

Parameters

→ *dbP*
: `DmOpenRef` to open database.

→ *index*
: Which record to delete.

Returns Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrReadOnly`
- `dmErrIndexOutOfRange`
- `dmErrRecordArchived`
- `dmErrRecordDeleted`
- `memErrInvalidParam`

Some releases may display a fatal error message instead of returning the error code.

Comments Marks the `delete` bit in the database header for the record and disposes of the record's data chunk. Does not remove the record entry from the database header, but simply sets the `localChunkID` of the record entry to `NULL`.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database.

For more information see "Accessing the PIM Application Databases" on page 2451.

See Also DmDetachRecord(), DmRemoveRecord(), DmArchiveRecord(), DmNewRecord()

DmDetachRecord Function

Purpose Detach and orphan a record from a database but don't delete the record's chunk.

Declared In DataMgr.h

Prototype

```
Err DmDetachRecord (DmOpenRef dbP, UInt16 index,
    MemHandle *oldHP)
```

Parameters

→ *dbP*
: DmOpenRef to open.

→ *index*
: Index of the record to detach.

↔ *oldHP*
: Pointer to return handle of the detached record.

Returns Returns errNone if no error, or one of the following if an error occurs:

- dmErrReadOnly
- dmErrIndexOutOfRange
- dmErrNotRecordDB
- memErrChunkLocked
- memErrInvalidParam

Some releases may display a fatal error message instead of returning the error code.

Comments This routine detaches a record from a database by removing its entry from the database header and returns the handle of the record's data chunk in *oldHP. Unlike DmDeleteRecord(), this routine removes its entry in the database header but it does not delete the actual record.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and

takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also DmAttachRecord(), DmRemoveRecord(), DmArchiveRecord(), DmDeleteRecord()

DmDetachResource Function

Purpose Detach a resource from a database and return the handle of the resource's data.

Declared In DataMgr.h

Prototype

```
Err DmDetachResource (DmOpenRef dbP,
   UInt16 index,
   MemHandle *oldHP)
```

Parameters

→ *dbP*
: DmOpenRef to open database.

→ *index*
: Index of resource to detach.

↔ *oldHP*
: Pointer to return handle of the detached record.

Returns Returns errNone if no error, or one of the following if an error occurs:

- dmErrReadOnly
- dmErrIndexOutOfRange
- dmErrCorruptDatabase
- memErrChunkLocked
- memErrInvalidParam

Some releases may display a fatal error message instead of returning the error code. All releases may display a fatal error message if the database is not a resource database.

Comments This routine detaches a resource from a database by removing its entry from the database header and returns the handle of the resource's data chunk in *oldHP.

See Also DmAttachResource(), DmRemoveResource()

DmFindDatabase Function

Purpose Return the database ID of a database by card number and name.

Declared In `DataMgr.h`

Prototype

```
LocalID DmFindDatabase (UInt16 cardNo,
   const Char *nameP)
```

Parameters

→ *cardNo*
Number of card to search.

→ *nameP*
Name of the database to look for.

Returns Returns the database ID. If the database can't be found, this function returns 0, and `DmGetLastErr()` returns an error code indicating the reason for failure.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmGetNextDatabaseByTypeCreator()`, `DmDatabaseInfo()`, `DmOpenDatabase()`

DmFindRecordByID Function

Purpose Return the index of the record with the given unique ID.

Declared In `DataMgr.h`

Prototype

```
Err DmFindRecordByID (DmOpenRef dbP,
   UInt32 uniqueID, UInt16 *indexP)
```

Parameters

→ *dbP*
Database access pointer.

→ *uniqueID*
Unique ID to search for.

← *indexP*
Return index.

Returns Returns 0 if found, otherwise `dmErrUniqueIDNotFound`. May display a fatal error message if the unique ID is invalid.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmQueryRecord()`, `DmGetRecord()`, `DmRecordInfo()`

DmFindResource Function

Purpose Search the given database for a resource by type and ID, or by pointer if it is non-`NULL`.

Declared In `DataMgr.h`

Prototype

```
UInt16 DmFindResource (DmOpenRef dbP,
   DmResType resType, DmResID resID,
   MemHandle resH)
```

Parameters

→ *`dbP`*
Open resource database access pointer.

→ *`resType`*
Type of resource to search for.

→ *`resID`*
ID of resource to search for.

→ *`resH`*
Pointer to locked resource, or `NULL`.

Returns Returns index of resource in resource database, or `dmInvalidRecIndex` if not found.

May display a fatal error message if the database is not a resource database.

Comments Use this routine to find a resource in a particular resource database by type and ID or by pointer. It is particularly useful when you want to search only one database for a resource and that database is not the topmost one.

IMPORTANT: This function searches for the resource only in the database you specify. If you pass a pointer to a base resource database, its overlay is *not* searched. To search both a base database and its overlay for a localized resource, use `DmGet1Resource()` instead of this function.

If `resH` is `NULL`, the resource is searched for by type and ID.

If `resH` is not `NULL`, `resType` and `resID` are ignored and the index of the given locked resource is returned.

Once the index of a resource is determined, it can be locked down and accessed by calling `DmGetResourceIndex()`.

See Also `DmGetResource()`, `DmSearchResource()`, `DmResourceInfo()`, `DmGetResourceIndex()`, `DmFindResourceType()`

DmFindResourceType Function

Purpose Search the given database for a resource by type and type index.

Declared In `DataMgr.h`

Prototype

```
UInt16 DmFindResourceType (DmOpenRef dbP,
    DmResType resType, UInt16 typeIndex)
```

Parameters

→ *dbP*
Open resource database access pointer.

→ *resType*
Type of resource to search for.

→ *typeIndex*
Index of given resource type.

Returns Index of resource in resource database, or `0xFFFF` if not found.

May display a fatal error message if the database is not a resource database.

Comments Use this routine to retrieve all the resources of a given type in a resource database. By starting at `typeIndex` 0 and incrementing until an error is returned, the total number of resources of a given type and the index of each of these resources can be determined.

Once the index of a resource is determined, it can be locked down and accessed by calling DmGetResourceIndex().

IMPORTANT: This function searches for resources only in the database you specify. If you pass a pointer to a base resource database, its overlay is *not* searched. To search both a base database and its overlay for a localized resource, use DmGet1Resource() instead of this function.

See Also DmGetResource(), DmSearchResource(), DmResourceInfo(), DmGetResourceIndex(), DmFindResource()

DmFindSortPosition Function

Purpose Returns where in a sorted list of records a given record would be located. Useful to find where to insert a record with DmAttachRecord(). Uses a binary search.

Declared In DataMgr.h

Prototype

```
UInt16 DmFindSortPosition (DmOpenRef dbP,
    void *newRecord,
    SortRecordInfoPtr newRecordInfo,
    DmComparF *compar, Int16 other)
```

Parameters

→ *dbP*
Database access pointer.

→ *newRecord*
Pointer to the new record.

→ *newRecordInfo*
Sort information about the new record. See SortRecordInfoType.

→ *compar*
Pointer to comparison function. See DmComparF().

→ *other*
Any value the application wants to pass to the comparison function. This parameter is often used to indicate a sort direction (ascending or descending).

Returns The position where the record should be inserted.

The position should be viewed as between the record returned and the record before it. Note that the return value may be one greater than the number of records.

Comments If `newRecord` has the same key as another record in the database, `DmFindSortPosition` assumes that `newRecord` should be inserted after that record. If there are several records with the same key, `newRecord` is inserted after all of them. For this reason, if you use `DmFindSortPosition` to search for the location of a record that you know is already in the database, you must subtract 1 from the result. (Be sure to check that the value is not 0.)

If there are deleted records in the database, `DmFindSortPosition` only works if those records are at the end of the database. `DmFindSortPosition` always assumes that a deleted record is greater than or equal to any other record.

Compatibility Implemented only if 2.0 New Feature Set is present.

If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmFindSortPositionV10()`

DmFindSortPositionV10 Function

Purpose Return where a record should be. Useful to find where to insert a record with `DmAttachRecord()`. Uses a binary search.

Declared In `DataMgr.h`

Prototype

```
UInt16 DmFindSortPositionV10 (DmOpenRef dbP,
    void *newRecord, DmComparF *compar,
    Int16 other)
```

Parameters

→ *dbP*
Database access pointer.

→ *newRecord*
Pointer to the new record.

→ *compar*
Pointer to comparison function. See DmComparF().

→ *other*
Any value the application wants to pass to the comparison function.

Returns Returns the position where the record should be inserted. The position should be viewed as between the record returned and the record before it. Note that the return value may be one greater than the number of records.

Comments If there are deleted records in the database, `DmFindSortPositionV10` only works if those records are at the end of the database. `DmFindSortPositionV10` always assumes that a deleted record is greater than or equal to any other record.

Compatibility This function corresponds to the 1.0 version of `DmFindSortPosition`.

See Also DmFindSortPosition(), DmQuickSort(), DmInsertionSort()

DmGetAppInfoID Function

Purpose Return the local ID of the application info block.

Declared In `DataMgr.h`

Prototype

```
LocalID DmGetAppInfoID (DmOpenRef dbP).
```

Parameters → *dbP*
Database access pointer.

Returns Returns local ID of the application info block. The application info block is an optional field that the database may use to store application-specific information about the database; if the database doesn't have an application info block, `DmGetAppInfoID` returns zero.

See Also DmDatabaseInfo(), DmOpenDatabase()

DmGetDatabase Function

Purpose Return the database header ID of a database by index and card number.

Declared In `DataMgr.h`

Prototype

```
LocalID DmGetDatabase (UInt16 cardNo,
   UInt16 index)
```

Parameters

→ *cardNo*
Card number of database.

→ *index*
Index of database.

Returns Returns the database ID, or 0 if an invalid parameter is passed.

Comments Call this routine to retrieve the database ID of a database by index. The index should range from 0 to `DmNumDatabases()`-1.

This routine is useful for getting a directory of all databases on a card. The databases returned may reside in either the ROM or the RAM. The order in which databases are returned is not fixed; therefore, you should not rely on receiving a list of databases in a particular order.

See Also `DmOpenDatabase()`, `DmNumDatabases()`, `DmDatabaseInfo()`, `DmDatabaseSize()`

DmGetDatabaseLockState Function

Purpose Return information about the number of locked and busy records in a database.

Declared In `DataMgr.h`

Prototype

```
void DmGetDatabaseLockState (DmOpenRef dbR,
   UInt8 *highest, UInt32 *count, UInt32 *busy)
```

Parameters

→ *dbR*
Database access pointer.

← *highest*
The highest lock count found for all of the records in the database. If a database has two records, one has a lock count

of 2 and one has a lock count of 1, the highest lock count is 2. Pass `NULL` for this parameter if you don't want to retrieve it.

← *`count`*
The number of records that have the lock count that is returned in the `highest` parameter. Pass `NULL` for this parameter if you don't want to retrieve it.

← *`busy`*
The number of records that have the busy bit set. Pass `NULL` for this parameter if you don't want to retrieve it.

Returns No return value. Returns all information in the parameters you pass.

Comments This function is intended to be used for debugging purposes. You can use it to obtain information about how many records are busy and how much locking occurs.

Compatibility Implemented only if 3.2 New Feature Set is present.

DmGetLastErr Function

Purpose Return error code from last data manager call.

Declared In `DataMgr.h`

Prototype

```
Err DmGetLastErr (void)
```

Parameters None.

Returns Error code from last unsuccessful data manager call.

Comments Use this routine to determine why a data manager call failed. In particular, calls like `DmGetRecord()` return 0 if unsuccessful, so calling `DmGetLastErr()` is the only way to determine why they failed.

Note that `DmGetLastErr` does not always reflect the error status of the last data manager call. Rather, it reflects the error status of data manager calls that don't return an error code. For some of those calls, the saved error code value is not set to 0 when the call is successful.

For example, if a call to `DmOpenDatabaseByTypeCreator()` returns `NULL` for database reference (that is, it fails),

`DmGetLastErr` returns something meaningful; otherwise, it returns the error value of some previous data manager call.

Only the following data manager functions currently affect the value returned by `DmGetLastErr`:

DmFindDatabase()	DmOpenDatabaseByTypeCreator()
DmOpenDatabase()	DmNewRecord()
DmQueryRecord()	DmGetRecord()
DmQueryNextInCategory()	DmPositionInCategory()
DmSeekRecordInCategory()	DmResizeRecord()
DmGetResource()	DmGet1Resource()
DmNewResource()	DmGetResourceIndex()
DmNewHandle()	DmOpenDBNoOverlay()
DmResizeResource()	

DmGetNextDatabaseByTypeCreator Function

Purpose Return a database header ID and card number given the type and/or creator. This routine searches all memory cards for a match.

Declared In `DataMgr.h`

Prototype

```
Err DmGetNextDatabaseByTypeCreator
    (Boolean newSearch,
    DmSearchStatePtr stateInfoP,
    UInt32 type, UInt32 creator,
    Boolean onlyLatestVers, UInt16 *cardNoP,
    LocalID *dbIDP)
```

Parameters

→ *newSearch*
: `true` if starting a new search.

↔ *stateInfoP*
: If `newSearch` is `false`, this must point to the same data used for the previous invocation.

→ *type*
: Type of database to search for, pass 0 as a wildcard.

→ *creator*
Creator of database to search for, pass 0 as a wildcard.

→ *onlyLatestVers*
If `true`, only the latest version of a database with a given type and creator is returned.

← *cardNoP*
On exit, the card number of the found database.

← *dbIDP*
Database local ID of the found database.

Returns Returns `errNone` if no error, or `dmErrCantFind` if no matches were found.

Comments You may need to call this function successively to discover all databases having a specified type/creator pair.

To start the search, pass `true` for `newSearch`. Allocate a `DmSearchStateType` structure and pass it as the `stateInfoP` parameter. `DmGetNextDatabaseByTypeCreator` stores private information in `stateInfoP` and uses it if the search is continued.

To continue a search where the previous one left off, pass `false` for `newSearch` and pass the same `stateInfoP` that you used during the previous call to this function.

You can pass `0` as a wildcard operator for the `type` or `creator` parameter to conduct searches of wider scope. If the `type` parameter is `0`, this routine can be called successively to return all databases of the given creator. If the `creator` parameter is `0`, this routine can be called successively to return all databases of the given type. You can also pass `0` as the value for both of these parameters to return all available databases without regard to type or creator.

Because databases are scattered freely throughout memory space, they are not returned in any particular order—any database matching the specified type/creator criteria can be returned.Thus, if the value of the `onlyLatestVers` parameter is `false`, this function may return a database which is not the most recent version matching the specified type/creator pair. To obtain only the latest version of a database matching the search criteria, set the value of the `onlyLatestVers` parameter to `true`.

When determining which is the latest version of the database, RAM databases are considered newer than ROM databases that have the same version number. Because of this, you can replace any ROM-based application with your own version of it. Also, a RAM database on card 1 is considered newer than a RAM database on card 0 if the version numbers are identical.

WARNING! Don't create or delete a database while using `DmGetNextDatabaseByTypeCreator` to iterate through the existing databases. This could cause databases to be skipped, or it could result in a given database being returned more than once.

Compatibility

In Palm OS version 3.1 and higher, if `onlyLatestVers` is `true`, you only receive one matching database for each type/creator pair. In version 3.0 and earlier, you could receive multiple matching databases if `onlyLatestVers` was `true`.

Note that the behavior is different only when you have specified a value for both `type` and `creator` and `onlyLatestVers` is `true`.

For example, suppose your application maintains three databases that all have the same type, creator, and version number and you write this code to process them in some way:

```
DmSearchStateType state;
Boolean latestVer;
UInt16 card;
LocalID currentDB = 0;

theErr = DmGetNextDatabaseByTypeCreator(true,
  &state, myType, myCreator, latestVer, &card,
  &currentDB);
while (!theErr && currentDB) {
  /* do something with currentDB */
  /* now get the next DB */
  theErr = DmGetNextDatabaseByTypeCreator(
    false, &state, myType, myCreator,
  vlatestVer, &card, &currentDB);
}
```

If `latestVer` is `false`, then your code will work the same on all versions of Palm OS and will return all three databases whose type and creator match those specified. If `latestVer` is `true`, this code

returns all three databases on Palm OS version 3.0 and earlier, but only returns one database on version 3.1 and higher. (Exactly which database it returns is unspecified.)

If you expect multiple databases to match your search criteria, make sure you call `DmGetNextDatabaseByTypeCreator` in one of the following ways to ensure that your code operates the same on all Palm OS versions:

- Set `onlyLatestVers` to `false` if you specify both a `type` and `creator`.
- Specify `0` for either the `type` or `creator` parameter (or both).

If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmGetDatabase()`, `DmFindDatabase()`, `DmDatabaseInfo()`, `DmOpenDatabaseByTypeCreator()`, `DmDatabaseSize()`

DmGetRecord Function

Purpose Return a handle to a record by index and mark the record busy.

Declared In `DataMgr.h`

Prototype

```
MemHandle DmGetRecord (DmOpenRef dbP,
    UInt16 index)
```

Parameters

→ *dbP*
: `DmOpenRef` to open database.

→ *index*
: Which record to retrieve.

Returns Returns a handle to record data. If another call to `DmGetRecord` for the same record is attempted before the record is released, `NULL` is returned and `DmGetLastErr()` returns an error code indicating the reason for failure.

Comments Returns a handle to given record and sets the `busy` bit for the record.

If the record is ROM-based (pointer accessed), this routine makes a fake handle to it and stores this handle in the `DmAccessType` structure.

`DmReleaseRecord()` should be called as soon as the caller finishes viewing or editing the record.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmSearchRecord()`, `DmFindRecordByID()`, `DmRecordInfo()`, `DmReleaseRecord()`, `DmQueryRecord()`

DmGetResource Function

Purpose Search all open resource databases and return a handle to a resource, given the resource type and ID.

Declared In `DataMgr.h`

Prototype

```
MemHandle DmGetResource (DmResType type,
    DmResID resID)
```

Parameters

→ *type*
The resource type.

→ *resID*
The resource ID.

Returns Handle to resource data. If the specified resource cannot be found, this function returns `NULL` and `DmGetLastErr()` returns an error code indicating the reason for failure.

Comments Searches all open resource databases starting with the most recently opened one for a resource of the given type and ID. If found, the resource handle is returned. The application should call `DmReleaseResource()` as soon as it finishes accessing the resource data. The resource handle is not locked by this function.

This function always returns the resource located in the overlay if any open overlay has a resource matching that type and ID. If there

is no overlay version of the resource, this function returns the resource from the base database.

See Also DmGet1Resource(), DmReleaseResource(), ResLoadConstant()

DmGetResourceIndex Function

Purpose Return a handle to a resource by index.

Declared In DataMgr.h

Prototype

```
MemHandle DmGetResourceIndex (DmOpenRef dbP,
   UInt16 index)
```

Parameters

→ *dbP*
Access pointer to open database.

→ *index*
Index of the resource whose handle you want.

Returns Handle to resource data. If the specified index is out of range, this function returns NULL and DmGetLastErr() returns an error code indicating the reason for failure.

May display a fatal error message if the database is not a resource database.

IMPORTANT: This function accesses the resource only in the database you specify. If you pass a pointer to a base resource database, its overlay is *not* accessed. Therefore, you should use care when using this function to access a potentially localized resource. You can use DmSearchResource() to obtain a pointer to the overlay database if the resource is localized; however, it's more convenient to use DmGetResource() or DmGet1Resource().

See Also DmFindResource(), DmFindResourceType(), DmSearchResource()

DmGet1Resource Function

Purpose Search the most recently opened resource database and return a handle to a resource given the resource type and ID.

Declared In `DataMgr.h`

Prototype

```
MemHandle DmGet1Resource (DmResType type,
    DmResID resID)
```

Parameters

→ *type*
The resource type.

→ *resID*
The resource ID.

Returns Handle to resource data. If unsuccessful, this function returns `NULL` and `DmGetLastErr()` returns an error code indicating the reason for failure.

Comments Searches the most recently opened resource database for a resource of the given type and ID. If the database has an overlay associated with it, the overlay is searched first, and then the base database is searched if the overlay does not contain the resource. If found, the resource handle is returned. The application should call `DmReleaseResource()` as soon as it finishes accessing the resource data. The resource handle is not locked by this function.

See Also `DmGetResource()`, `DmReleaseResource()`, `ResLoadConstant()`

DmInsertionSort Function

Purpose Sort records in a database.

Declared In `DataMgr.h`

Prototype

```
Err DmInsertionSort (DmOpenRef dbR,
    DmComparF *compar, Int16 other)
```

Parameters

→ *dbR*
Database access pointer.

→ *compar*
Comparison function. See `DmComparF()`.

→ *other*
Any value the application wants to pass to the comparison function. This parameter is often used to indicate a sort direction (ascending or descending).

Returns Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrReadOnly`
- `dmErrNotRecordDB`

Some releases may display a fatal error message instead of returning the error code.

Comments Deleted records are placed last in any order. All others are sorted according to the passed comparison function. Only records which are out of order move. Moved records are moved to the end of the range of equal records. If a large number of records are being sorted, try to use the quick sort.

The following insertion-sort algorithm is used: Starting with the second record, each record is compared to the preceding record. Each record not greater than the last is inserted into sorted position within those already sorted. A binary insertion is performed. A moved record is inserted after any other equal records.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmQuickSort()`

DmMoveCategory Function

Purpose Move all records in a category to another category.

Declared In `DataMgr.h`

Prototype

```
Err DmMoveCategory (DmOpenRef dbP,
   UInt16 toCategory, UInt16 fromCategory,
   Boolean dirty)
```

Parameters

→ *dbP*
: `DmOpenRef` to open database.

→ *toCategory*
: Category to which the records should be added.

→ *fromCategory*
: Category from which to remove records.

→ *dirty*
: If `true`, set the dirty bit.

Returns Returns 0 if successful, or `dmErrReadOnly` if the database is in read-only mode. Some releases may display a fatal error message instead of returning the error code.

Comments If `dirty` is `true`, the moved records are marked as dirty.

The `toCategory` and `fromCategory` parameters hold category index values. You can learn which category a record is in with the `DmRecordInfo()` call and use that value in this function. For example, the following code, ensures that the records `rec1` and `rec2` are in the same category:

```
DmOpenRef myDB;
UInt16 rec1, rec2;
UInt16 rec1Attr, rec2Attr;
UInt16 category1, category2;

DmRecordInfo (myDB, rec1, &rec1Attr, NULL,
  NULL);
category1 = rec1Attr & dmRecAttrCategoryMask;
DmRecordInfo(myDB, rec2, &rec2Attr, NULL,
  NULL);
category2 = rec2Attr & dmRecAttrCategoryMask;
if (category1 != category2)
  DmMoveCategory(myDB, category1, category2,
    true);
```

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, `sysErrNotAllowed` is returned. For more information see "Accessing the PIM Application Databases" on page 2451.

DmMoveRecord Function

Purpose Move a record from one index to another.

Declared In `DataMgr.h`

Prototype

```
Err DmMoveRecord (DmOpenRef dbP, UInt16 from,
    UInt16 to)
```

Parameters

→ *dbP*
: `DmOpenRef` to open database.

→ *from*
: Index of record to move.

→ *to*
: Where to move the record.

Returns Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrReadOnly`
- `dmErrIndexOutOfRange`
- `dmErrNotRecordDB`
- `dmErrMemError`
- `memErrInvalidParam`
- `memErrChunkLocked`

Some releases may display a fatal error message instead of returning the error code.

Comments Insert the record at the `to` index and move other records down. The `to` position should be viewed as an insertion position. This value may be one greater than the index of the last record in the database. In cases where `to` is greater than `from`, the new index of the record becomes `to`–1 after the move is complete.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and

takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

DmNewHandle Function

Purpose Attempt to allocate a new chunk in the same data heap or card as the database header of the passed database access pointer. If there is not enough space in that data heap, try other heaps.

Declared In `DataMgr.h`

Prototype

```
MemHandle DmNewHandle (DmOpenRef dbP,
   UInt32 size)
```

Parameters

→ *dbP*
: `DmOpenRef` to open database.

→ *size*
: Size of new handle.

Returns Returns the `chunkID` of new chunk. If an error occurs, returns 0, and `DmGetLastErr()` returns an error code indicating the reason for failure.

Comments Allocates a new handle of the given size. Ensures that the new handle is in the same memory card as the given database. This guarantees that you can attach the handle to the database as a record to obtain and save its `LocalID` in the `appInfoID` or `sortInfoID` fields of the header.

The handle should be attached to a database as soon as possible. If it is not attached to a database and the application crashes, the memory used by the new handle is unavailable until the next soft reset.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

DmNewRecord Function

Purpose Return a handle to a new record in the database and mark the record busy.

Declared In `DataMgr.h`

Prototype

```
MemHandle DmNewRecord (DmOpenRef dbP,
    UInt16 *atP,
    UInt32 size)
```

Parameters

→ *dbP*
: `DmOpenRef` to open database.

↔ *atP*
: Pointer to index where new record should be placed. Specify the value `dmMaxRecordIndex` to add the record to the end of the database.

→ *size*
: Size of new record.

Returns Handle to record data. If an error occurs, this function returns 0 and `DmGetLastErr()` returns an error code indicating the reason for failure.

Some releases may display a fatal error message if the database is opened in read-only mode or it is a resource database.

Comments Allocates a new record of the given size, and returns a handle to the record data. The parameter `atP` points to an index variable. The new record is inserted at index `*atP` and all record indices that follow are shifted down. If `*atP` is greater than the number of records currently in the database, the new record is appended to the end and its index is returned in `*atP`.

Both the `busy` and `dirty` bits are set for the new record and a unique ID is automatically created.

`DmReleaseRecord()` should be called as soon as the caller finishes viewing or editing the record.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database.

For more information see "Accessing the PIM Application Databases" on page 2451.

See Also DmAttachRecord(), DmRemoveRecord(), DmDeleteRecord()

DmNewResource Function

Purpose Allocate and add a new resource to a resource database.

Declared In DataMgr.h

Prototype

```
MemHandle DmNewResource (DmOpenRef dbP,
   DmResType resType, DmResID resID, UInt32 size)
```

Parameters

→ *dbP*
DmOpenRef to open database.

→ *resType*
Type of the new resource.

→ *resID*
ID of the new resource.

→ *size*
Desired size of the new resource.

Returns Returns a handle to the new resource. If an error occurs, this function returns NULL and DmGetLastErr() returns an error code indicating the reason for failure.

May display a fatal error message if the database is not a resource database.

Comments Allocates a memory chunk for a new resource and adds it to the given resource database. The new resource has the given type and ID. If successful, the application should call DmReleaseResource() as soon as it finishes initializing the resource.

See Also DmAttachResource(), DmRemoveResource()

DmNextOpenDatabase Function

Purpose Return `DmOpenRef` to next open database for the current task.

Declared In `DataMgr.h`

Prototype
```
DmOpenRef DmNextOpenDatabase (DmOpenRef currentP)
```

Parameters → *currentP*
Current database access pointer or `NULL`.

Returns `DmOpenRef` to next open database, or `NULL` if there are no more.

Comments Call this routine successively to get the `DmOpenRefs` of all open databases. Pass `NULL` for `currentP` to get the first one. Applications don't usually call this function, but is useful for system information.

See Also `DmOpenDatabaseInfo()`, `DmDatabaseInfo()`

DmNextOpenResDatabase Function

Purpose Return access pointer to next open resource database in the search chain.

Declared In `DataMgr.h`

Prototype
```
DmOpenRef DmNextOpenResDatabase (DmOpenRef dbP)
```

Parameters → *dbP*
Database reference, or 0 to start search from the top.

Returns Pointer to next open resource database.

Comments Returns pointer to next open resource database. To get a pointer to the first one in the search chain, pass `NULL` for `dbP`. This is the database that is searched when `DmGet1Resource()` is called.

If you use this function to access a resource database that might have an overlay associated with it, be careful how you use the result. The `DmOpenRef` returned by this function is a pointer to the overlay database, not the base database. If you subsequently pass this pointer to `DmFindResource()`, you'll receive a handle to the overlaid resource. If you're searching for a resource that is found only in the base, you won't find it. Instead, always use `DmGetResource()` or `DmGet1Resource()` to obtain a resource.

Both of those functions search both the overlay databases and their associated base databases.

DmNumDatabases Function

Purpose Determine how many databases reside on a memory card.

Declared In `DataMgr.h`

Prototype
```
UInt16 DmNumDatabases (UInt16 cardNo)
```

Parameters → *cardNo*
Number of the card to check.

Returns The number of databases found.

Comments This routine is helpful for getting a directory of all databases on a card. The routine `DmGetDatabase()` accepts an index from 0 to `DmNumDatabases()` -1 and returns a database ID by index.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmGetDatabase()`

DmNumRecords Function

Purpose Return the number of records in a database.

Declared In `DataMgr.h`

Prototype
```
UInt16 DmNumRecords (DmOpenRef dbP)
```

Parameters → *dbP*
`DmOpenRef` to open database.

Returns The number of records in a database.

Comments Records that have that have the `deleted` bit set (that is, records that will be deleted during the next synchronization because the user has marked them deleted) are included in the count. If you want to exclude these records from your count, use

DmNumRecordsInCategory() and pass dmAllCategories as the category.

See Also DmNumRecordsInCategory(), DmRecordInfo(), DmSetRecordInfo()

DmNumRecordsInCategory Function

Purpose Return the number of records of a specified category in a database.

Declared In DataMgr.h

Prototype

```
UInt16 DmNumRecordsInCategory (DmOpenRef dbP,
   UInt16 category)
```

Parameters

→ *dbP*
DmOpenRef to open database.

→ *category*
Category index.

Returns The number of records in the category.

Comments Because this function must examine all records in the database, it can be slow to return, especially when called on a large database.

Records that have the deleted bit set are not counted, and if the user has specified to hide or mask private records, private records are not counted either.

You can use the DmRecordInfo() call to obtain a category index from a given record. For example:

```
DmOpenRef myDB;
UInt16 record, attr, category, total;

DmRecordInfo(myDB, record, &attr, NULL, NULL);
category = attr & dmRecAttrCategoryMask;
total = DmNumRecordsInCategory(myDB, category);
```

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database.

For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmNumRecords()`, `DmQueryNextInCategory()`, `DmPositionInCategory()`, `DmSeekRecordInCategory()`, `DmMoveCategory()`

DmNumResources Function

Purpose Return the total number of resources in a given resource database.

Declared In `DataMgr.h`

Prototype

```
UInt16 DmNumResources (DmOpenRef dbP)
```

Parameters

→ *dbP*
`DmOpenRef` to open database.

Returns The total number of resources in the given database.

May display a fatal error message if the database is not a resource database.

Comments `DmNumResources` only counts the resources in the database indicated by the `DmOpenRef` parameter. If the database is a resource database that has an overlay associated with it, this function only returns the number of resources in the base database, not in the overlay.

DmOpenDatabase Function

Purpose Open a database and return a reference to it. If the database is a resource database, also open its overlay for the current locale.

Declared In `DataMgr.h`

Prototype

```
DmOpenRef DmOpenDatabase (UInt16 cardNo,
   LocalID dbID, UInt16 mode)
```

Parameters

→ *cardNo*
Card number database resides on.

→ *dbID*
The database ID of the database.

→ *mode*
Which mode to open database in (see "Open Mode Constants").

Returns Returns `DmOpenRef` to open database. May display a fatal error message if the database parameter is `NULL` or is an invalid ID. On all other errors, this function returns 0 and `DmGetLastErr()` returns an error code indicating the reason for failure.

Comments Call this routine to open a database for reading or writing.

This routine returns a `DmOpenRef` which must be used to access particular records in a database. If unsuccessful, 0 is returned and the cause of the error can be determined by calling `DmGetLastErr()`.

When you use this routine to open a resource database in read-only mode, it also opens the overlay associated with this database for the current locale, if it exists. (The function `OmGetCurrentLocale()` returns the current locale.) Overlays are resource databases typically used to localize applications, shared libraries, and panels. They have the same creator as the base database, a type of `'ovly'` (symbolically named `omOverlayDBType`), and contain resources with the same IDs and types as the resources in the base database. When you request a resource from the database using `DmGetResource()` or `DmGet1Resource()`, the overlay is searched first. If the overlay contains a resource for the given ID, it is returned. If not, the resource from the base database is returned.

The `DmOpenRef` returned by this function is the pointer to the base database, not to the overlay database, so care should be taken when passing this pointer to functions such as `DmFindResource()` because this circumvents the overlay.

It's possible to create a "stripped" base resource database, one that does not contain any user interface resources. `DmOpenDatabase` only opens a stripped database if its corresponding overlay exists. If the overlay does not exist or if the overlay doesn't match the resource database, `DmOpenDatabase` returns `NULL` and `DmGetLastErr()` returns the error code `omErrBaseRequiresOverlay`.

If you open a resource database in a writable mode, the associated overlay is not opened. If you make changes to the resource

database, the overlay database is invalidated if those changes affect any resources that are also in the overlay. This means that on future occasions where you open the resource database in read-only mode, the overlay will not be opened because Palm OS considers it to be invalid.

TIP: If you want to prevent your resource database from being overlaid, include an `'xprf'` resource (symbolically named `sysResTExtPrefs`) in the database with the ID 0 (`sysResIDExtPrefs`) and set its `disableOverlays` flag. This resource is defined in `UIResources.r`.

Compatibility

Overlay support is only available if 3.5 New Feature Set is present. On earlier releases, this function opens resource databases without looking for an associated overlay.

If 4.0 New Feature Set is present, when `DmOpenDatabase` attempts to open a stripped resource database and cannot find an overlay for it, it searches for an overlay matching the default locale if the system locale is different from the default locale.

If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also

`DmCloseDatabase()`, `DmCreateDatabase()`, `DmFindDatabase()`, `DmOpenDatabaseByTypeCreator()`, `DmDeleteDatabase()`, `DmOpenDBNoOverlay()`

DmOpenDatabaseByTypeCreator Function

Purpose Open the most recent revision of a database with the given type and creator. If the database is a resource database, also open its overlay for the current locale.

Declared In `DataMgr.h`

Prototype

```
DmOpenRef DmOpenDatabaseByTypeCreator
    (UInt32 type, UInt32 creator, UInt16 mode)
```

Parameters

→ *type*
Type of database.

→ *creator*
Creator of database.

→ *mode*
Which mode to open database in (see "Open Mode Constants").

Returns `DmOpenRef` to open database. If the database couldn't be found, this function returns 0 and `DmGetLastErr()` returns an error code indicating the reason for failure.

Comments If you use this routine to open a resource database in read-only mode, it also opens the overlay associated with this database for the current locale. See `DmOpenDatabase()` for more information on overlays and resource databases.

Compatibility Overlay support is only available if 3.5 New Feature Set is present. On earlier releases, this function opens resource databases without looking for an associated overlay.

If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmCreateDatabase()`, `DmOpenDatabase()`, `DmOpenDatabaseInfo()`, `DmCloseDatabase()`, `DmOpenDBNoOverlay()`

DmOpenDatabaseInfo Function

Purpose Retrieve information about an open database.

Declared In `DataMgr.h`

Prototype

```
Err DmOpenDatabaseInfo (DmOpenRef dbP,
   LocalID *dbIDP, UInt16 *openCountP,
   UInt16 *modeP,
   UInt16 *cardNoP, Boolean *resDBP)
```

Parameters

→ *dbP*
: `DmOpenRef` to open database.

← *dbIDP*
: The ID of the database. Pass `NULL` for this parameter if you don't want to retrieve this information.

← *openCountP*
: The number of applications that have this database open. Pass `NULL` for this parameter if you don't want to retrieve this information.

← *modeP*
: The mode used to open the database (see "Open Mode Constants"). Pass `NULL` for this parameter if you don't want to retrieve this information.

← *cardNoP*
: The number of the card on which this database resides. Pass `NULL` for this parameter if you don't want to retrieve this information.

← *resDBP*
: If `true` upon return, the database is a resource database, `false` otherwise. Pass `NULL` for this parameter if you don't want to retrieve this information.

Returns Returns `errNone` if no error.

See Also `DmDatabaseInfo()`

DmOpenDBNoOverlay Function

Purpose Open a database and return a reference to it.

Declared In `DataMgr.h`

Prototype

```
DmOpenRef DmOpenDBNoOverlay (UInt16 cardNo,
   LocalID dbID, UInt16 mode)
```

Parameters

→ *cardNo*
Card number database resides on.

→ *dbID*
The database ID of the database.

→ *mode*
Which mode to open database in (see "Open Mode Constants").

Returns `DmOpenRef` to open database. May display a fatal error message if the database parameter is `NULL`. On all other errors, this function returns 0 and `DmGetLastErr()` returns an error code indicating the reason for failure.

Comments Call this routine to open a database for reading or writing, while ignoring any overlay databases that might be associated with it.

This routine returns a `DmOpenRef` which must be used to access particular records in a database. If unsuccessful, 0 is returned and the cause of the error can be determined by calling `DmGetLastErr()`.

Compatibility Implemented only if 3.5 New Feature Set is present.

If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmCloseDatabase()`, `DmCreateDatabase()`, `DmFindDatabase()`, `DmOpenDatabaseByTypeCreator()`, `DmDeleteDatabase()`, `DmOpenDatabase()`

DmPositionInCategory Function

Purpose Return a position of a record within the specified category.

Declared In `DataMgr.h`

Prototype

```
UInt16 DmPositionInCategory (DmOpenRef dbP,
   UInt16 index, UInt16 category)
```

Parameters

→ *dbP*
: `DmOpenRef` to open database.

→ *index*
: Index of the record.

→ *category*
: Index of category to search.

Returns Returns the position (zero-based). If the specified index is out of range, this function returns 0 and `DmGetLastErr()` returns an error code indicating the reason for failure. Note that this means a 0 return value might indicate either success or failure. If this function returns 0 and `DmGetLastErr` returns `errNone`, the return value indicates that this is the first record in the category.

Comments Because this function must examine all records up to the current record, it can be slow to return, especially when called on a large database.

Records that have the `deleted` bit set are ignored, and if the user has specified that private records should be hidden or masked, private records are ignored as well.

If the record is ROM-based (pointer accessed) this routine makes a fake handle to it and stores this handle in the `DmAccessType` structure.

To learn which category a record is in, use `DmRecordInfo()`.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmQueryNextInCategory()`, `DmSeekRecordInCategory()`, `DmMoveCategory()`

DmQueryNextInCategory Function

Purpose Return a handle to the next record in the specified category for reading only (does not set the `busy` bit).

Declared In `DataMgr.h`

Prototype

```
MemHandle DmQueryNextInCategory (DmOpenRef dbP,
    UInt16 *indexP, UInt16 category)
```

Parameters

→ *dbP*
: `DmOpenRef` to open database.

↔ *indexP*
: Index of a known record (often retrieved with `DmPositionInCategory()`). If a "next" record is found, this index is updated to indicate that record.

→ *category*
: Index of category to query, or `dmAllCategories` to find the next record in any category.

Returns Returns a handle to the record following a known record, along with the index of that record. If a record couldn't be found, this function returns `NULL`, and `DmGetLastErr()` returns an error code indicating the reason for failure.

Comments This function begins searching the database from the record at `*indexP` for a record that is in the specified category. If the record at `*indexP` belongs to that category, then a handle to it is returned. If not, the function continues searching until it finds a record in the category.

Records that have the `deleted` bit set are skipped, and if the user has specified that private records should be hidden or masked, private records are skipped as well.

Thus, if you want to find the next record in the category after the one you have an index for, increment the index value before calling this function. For example:

```
DmOpenRef myDB;
UInt16 record, attr, category, pos;
MemHandle newRecH;

DmRecordInfo(myDB, record, &attr, NULL, NULL);
category = attr & dmRecAttrCategoryMask;
pos = DmPositionInCategory(myDB, record,
```

```
  category);
pos++;
newRecH = DmQueryNextInCategory(myDB, &pos, category);
```

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmNumRecordsInCategory()`, `DmPositionInCategory()`, `DmSeekRecordInCategory()`

DmQueryRecord Function

Purpose Return a handle to a record for reading only (does not set the `busy` bit).

Declared In `DataMgr.h`

Prototype

```
MemHandle DmQueryRecord (DmOpenRef dbP,
    UInt16 index)
```

Parameters

→ *dbP*
: `DmOpenRef` to open database.

→ *index*
: Which record to retrieve.

Returns Returns a record handle. If an error occurs, this function returns `NULL`, and `DmGetLastErr()` returns an error code indicating the reason for failure.

Some releases may display a fatal error message if the specified index is out of range.

Comments Returns a handle to the given record. Use this routine only when viewing the record. This routine successfully returns a handle to the record even if the record is busy.

If the record is ROM-based (pointer accessed) this routine returns the fake handle to it.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and

takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

DmQuickSort Function

Purpose Sort records in a database.

Declared In `DataMgr.h`

Prototype

```
Err DmQuickSort (DmOpenRef dbP,
    DmComparF *compar,
    Int16 other)
```

Parameters

→ *dbP*
Database access pointer.

→ *compar*
Comparison function. See `DmComparF()`.

→ *other*
Any value the application wants to pass to the comparison function. This parameter is often used to indicate a sort direction (ascending or descending).

Returns Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrReadOnly`
- `dmErrNotRecordDB`

Some releases may display a fatal error message instead of returning the error code.

Comments Deleted records are placed last in any order. All others are sorted according to the passed comparison function.

After `DmQuickSort` returns, equal database records do not have a consistent order. That is, if `DmQuickSort` is passed two equal records, their resulting order is unpredictable. To prevent records that contain the same data from being rearranged in an unpredictable order, pass the record's unique ID to the comparison function (using the `SortRecordInfoType` structure).

`DmQuickSort` contains its own stack to limit uncontrolled recursion. When the stack is full `DmQuickSort` instead performs an

insertion sort. An insertion sort is also performed when the number of records is low, avoiding the noticeable overhead of a quick sort with a small number of records. Finally, if the records seem mostly sorted an insertion sort is performed to move only those records that need moving.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmFindSortPositionV10()`, `DmInsertionSort()`

DmRecordInfo Function

Purpose Retrieve the record information as stored in the database header.

Declared In `DataMgr.h`

Prototype

```
Err DmRecordInfo (DmOpenRef dbP, UInt16 index,
   UInt16 *attrP, UInt32 *uniqueIDP,
   LocalID *chunkIDP)
```

Parameters

→ *dbP*
: `DmOpenRef` to open database.

→ *index*
: Index of the record.

← *attrP*
: The record's attributes. See "Record Attribute Constants." Pass `NULL` for this parameter if you don't want to retrieve this value.

← *uniqueIDP*
: The record's unique ID. Pass `NULL` for this parameter if you don't want to retrieve this value.

← *chunkIDP*
: The record's local ID. Pass `NULL` for this parameter if you don't want to retrieve this value.

Returns Returns `errNone` if no error or `dmErrIndexOutOfRange` if the specified record can't be found. Some releases may display a fatal error message instead of returning the error code.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmNumRecords()`, `DmSetRecordInfo()`, `DmQueryNextInCategory()`

DmReleaseRecord Function

Purpose Clear the `busy` bit for the given record and set the `dirty` bit if dirty is `true`.

Declared In `DataMgr.h`

Prototype

```
Err DmReleaseRecord (DmOpenRef dbP, UInt16 index,
    Boolean dirty)
```

Parameters

→ *dbP*
: `DmOpenRef` to open database.

→ *index*
: The record to unlock.

→ *dirty*
: If `true`, set the `dirty` bit.

Returns Returns `errNone` if no error, or `dmErrIndexOutOfRange` if the specified index is out of range. Some releases may display a fatal error message instead of returning the error code.

Comments Call this routine when you finish modifying or reading a record that you've called `DmGetRecord()` on or created using `DmNewRecord()`.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmGetRecord()`

DmReleaseResource Function

Purpose Release a resource acquired with DmGetResource().

Declared In `DataMgr.h`

Prototype
```
Err DmReleaseResource (MemHandle resourceH)
```

Parameters
→ *resourceH*
Handle to resource.

Returns Returns `errNone` if no error.

Comments Marks a resource as being no longer needed by the application.

See Also DmGet1Resource(), DmGetResource()

DmRemoveRecord Function

Purpose Remove a record from a database and dispose of its data chunk.

Declared In `DataMgr.h`

Prototype
```
Err DmRemoveRecord (DmOpenRef dbP, UInt16 index)
```

Parameters
→ *dbP*
`DmOpenRef` to open database.

→ *index*
Index of the record to remove.

Returns Returns `errNone` if no error, or one of the following if an error occurs:

- dmErrReadOnly
- dmErrIndexOutOfRange
- dmErrNotRecordDB
- memErrChunkLocked
- memErrInvalidParam

Some releases may display a fatal error message instead of returning the error code.

Comments Disposes of a the record's data chunk and removes the record's entry from the database header. `DmRemoveRecord` should only be used for newly-created records that have just been deleted or records that have never been sync'ed.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmDetachRecord()`, `DmDeleteRecord()`, `DmArchiveRecord()`, `DmNewRecord()`

DmRemoveResource Function

Purpose Delete a resource from a resource database.

Declared In `DataMgr.h`

Prototype

```
Err DmRemoveResource (DmOpenRef dbP,
   UInt16 index)
```

Parameters

→ *dbP*
: `DmOpenRef` to open database.

→ *index*
: Index of resource to delete.

Returns Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrCorruptDatabase`
- `dmErrIndexOutOfRange`
- `dmErrReadOnly`
- `memErrChunkLocked`
- `memErrInvalidParam`
- `memErrNotEnoughSpace`

May display a fatal error message if the database is not a resource database.

Comments This routine disposes of the memory manager chunk that holds the given resource and removes its entry from the database header.

See Also `DmDetachResource()`, `DmRemoveResource()`, `DmAttachResource()`

DmRemoveSecretRecords Function

Purpose Remove all secret records.

Declared In `DataMgr.h`

Prototype

```
Err DmRemoveSecretRecords (DmOpenRef dbP)
```

Parameters

→ *dbP*
`DmOpenRef` to open database.

Returns Returns `errNone` if no error, or one of the following if an error occurs:

- `dmErrReadOnly`
- `dmErrNotRecordDB`

Some releases may display a fatal error message instead of returning the error code.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, `sysErrNotAllowed` is returned. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also `DmRemoveRecord()`, `DmRecordInfo()`, `DmSetRecordInfo()`

DmResizeRecord Function

Purpose Resize a record by index.

Declared In `DataMgr.h`

Prototype

```
MemHandle DmResizeRecord (DmOpenRef dbP,
   UInt16 index, UInt32 newSize)
```

Parameters

→ *dbP*
`DmOpenRef` to open database.

→ *index*
Which record to retrieve.

→ *newSize*
New size of record.

Returns Handle to resized record. Returns `NULL` if there is not enough space to resize the record, and `DmGetLastErr()` returns an error code

indicating the reason for failure. Some releases may display a fatal error message instead of returning the error code.

Comments This routine reallocates the record in another heap of the same memory card if the current heap is not big enough. If this happens, the handle changes, so be sure to use the returned handle to access the resized record.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

DmResizeResource Function

Purpose Resize a resource and return the new handle.

Declared In `DataMgr.h`

Prototype

```
MemHandle DmResizeResource (MemHandle resourceH,
    UInt32 newSize)
```

Parameters

→ *resourceH*
Handle to resource.

→ *newSize*
Desired new size of resource.

Returns Returns a handle to newly sized resource. Returns `NULL` if there is not enough space to resize the resource, and `DmGetLastErr()` returns an error code indicating the reason for failure. Some releases may display a fatal error message instead of returning the error code.

Comments Resizes the resource and returns a new handle. If necessary in order to grow the resource, this routine will reallocate it in another heap on the same memory card that it is currently in.

The handle may change if the resource had to be reallocated in a different data heap because there was not enough space in its present data heap.

DmResourceInfo Function

Purpose Retrieve information on a given resource.

Declared In `DataMgr.h`

Prototype

```
Err DmResourceInfo (DmOpenRef dbP, UInt16 index,
   DmResType *resTypeP, DmResID *resIDP,
   LocalID *chunkLocalIDP)
```

Parameters

→ *dbP*
: `DmOpenRef` to open database.

→ *index*
: Index of resource to get info on.

← *resTypeP*
: The resource type. Pass `NULL` if you don't want to retrieve this information.

← *resIDP*
: The resource ID. Pass `NULL` if you don't want to retrieve this information.

← *chunkLocalIDP*
: The memory manager local ID of the resource data. Pass `NULL` if you don't want to retrieve this information.

Returns Returns `errNone` if no error or `dmErrIndexOutOfRange` if an error occurred. May display a fatal error message if the database is not a resource database.

Comments If `dbP` is a pointer to a base resource database, the information returned is about the resource from that database alone; this function ignores any associated overlay.

See Also `DmGetResource()`, `DmGet1Resource()`, `DmSetResourceInfo()`, `DmFindResource()`, `DmFindResourceType()`

DmSearchRecord Function

Purpose Search all open record databases for a record with the handle passed.

Declared In `DataMgr.h`

Prototype

```
UInt16 DmSearchRecord (MemHandle recH,
    DmOpenRef *dbPP)
```

Parameters

→ *recH*
Record handle.

← *dbPP*
The database that contains the record `recH`.

Returns Returns the index of the record and database access pointer; if not found, returns -1 and `*dbPP` is 0.

See Also `DmGetRecord()`, `DmFindRecordByID()`, `DmRecordInfo()`

DmSearchResource Function

Purpose Search all open resource databases for a resource by type and ID, or by pointer if it is non-`NULL`.

Declared In `DataMgr.h`

Prototype

```
UInt16 DmSearchResource (DmResType resType,
    DmResID resID, MemHandle resH,
    DmOpenRef *dbPP)
```

Parameters

→ *resType*
Type of resource to search for.

→ *resID*
ID of resource to search for.

→ *resH*
Pointer to locked resource, or `NULL`.

← *dbPP*
The resource database that contains the specified resource.

Returns Returns the index of the resource, stores `DmOpenRef` in `dbPP`.

Comments This routine can be used to find a resource in all open resource databases by type and ID or by pointer. If `resH` is `NULL`, the resource is searched for by type and ID. If `resH` is not `NULL`,

`resType` and `resID` is ignored and the index of the resource handle is returned. On return, `*dbPP` contains the access pointer of the resource database that the resource was eventually found in. Once the index of a resource is determined, it can be locked down and accessed by calling `DmGetResourceIndex()`.

If any of the open databases are overlaid, this function finds and returns the localized version of the resource when searching by type and creator. In this case, the `dbPP` return value is a pointer to the overlay database, not the base resource database.

See Also `DmGetResource()`, `DmFindResourceType()`, `DmResourceInfo()`, `DmFindResource()`

DmSeekRecordInCategory Function

Purpose Return the index of the record nearest the offset from the passed record index whose category matches the passed category. (The `offset` parameter indicates the number of records to move forward or backward.)

Declared In `DataMgr.h`

Prototype

```
Err DmSeekRecordInCategory (DmOpenRef dbP,
   UInt16 *indexP, UInt16 offset,
   Int16 direction,
   UInt16 category)
```

Parameters

→ *dbP*
: `DmOpenRef` to open database.

↔ *index*
: The index to start the search at. Upon return, contains the index of the record at `offset` from the index that you passed in.

→ *offset*
: Offset of the passed record index. This must be a positive number; use `dmSeekBackward` for the `direction` parameter to search backwards.

→ *direction*
: Must be either `dmSeekForward` or `dmSeekBackward`.

→ *category*
Category index.

Returns Returns `errNone` if no error; returns `dmErrIndexOutOfRange` or `dmErrSeekFailed` if an error occurred.

Comments `DmSeekRecordInCategory` searches for a record in the specified category. The search begins with the record at `index`. When it finds a record in the specified category, it decrements the `offset` parameter and continues searching until a match is found and `offset` is 0.

Because of this, if you use `DmSeekRecordInCategory` to find the nearest matching record in a particular category, you must pass different `offset` parameters if the starting record is in the category than if it isn't. If the record at `index` is in the category, then you must pass an `offset` of 1 to find the next record in the category because the comparison is performed before the `index` value changes. If the record at `index` isn't in the category, you must pass an `offset` of 0 to find the next record in the category. In this case, an `offset` of 1 skips the first matching record.

Records that have the `deleted` bit set are ignored, and if the user has specified that private records should be hidden or masked, private records are ignored as well.

See Also `DmNumRecordsInCategory()`, `DmQueryNextInCategory()`, `DmPositionInCategory()`, `DmMoveCategory()`

DmSet Function

Purpose Write a specified value into a section of a record. This function also checks the validity of the pointer for the record and makes sure the writing of the record information doesn't exceed the bounds of the record.

Declared In `DataMgr.h`

Prototype

```
Err DmSet (void *recordP, UInt32 offset,
   UInt32 bytes, UInt8 value)
```

Parameters → *recordP*
Pointer to locked data record (chunk pointer).

→ *offset*
Offset within record to start writing.

→ *bytes*
Number of bytes to write.

→ *value*
Byte value to write.

Returns Returns `errNone` if no error. May display a fatal error message if the record pointer is invalid or the function overwrites the record.

Comments Must be used to write to data manager records because the data storage area is write-protected.

See Also `DmWrite()`

DmSetDatabaseInfo Function

Purpose Set information about a database.

Declared In `DataMgr.h`

Prototype

```
Err DmSetDatabaseInfo (UInt16 cardNo,
   LocalID dbID, const Char *nameP,
   UInt16 *attributesP, UInt16 *versionP,
   UInt32 *crDateP, UInt32 *modDateP,
   UInt32 *bckUpDateP, UInt32 *modNumP,
   LocalID *appInfoIDP, LocalID *sortInfoIDP,
   UInt32 *typeP, UInt32 *creatorP)
```

Parameters

→ *cardNo*
Card number the database resides on.

→ *dbID*
Database ID of the database.

→ *nameP*
Pointer to the new name of the database, or `NULL`. A database name can be up to 32 ASCII bytes long, including the null terminator (as specified by `dmDBNameLength`). Database names must use only 7-bit ASCII characters (0x20 through 0x7E).

→ *attributesP*
Pointer to new attributes variable, or `NULL`. See "Database Attribute Constants" for a list of possible values.

→ *versionP*
Pointer to new version, or `NULL`.

→ *crDateP*
Pointer to new creation date variable, or `NULL`. Specify the value as a number of seconds since Jan. 1, 1904.

→ *modDateP*
Pointer to new modification date variable, or `NULL`. Specify the value as a number of seconds since Jan. 1, 1904.

→ *bckUpDateP*
Pointer to new backup date variable, or `NULL`. Specify the value as a number of seconds since Jan. 1, 1904.

→ *modNumP*
Pointer to new modification number variable, or `NULL`.

→ *appInfoIDP*
Pointer to new `appInfoID` variable, or `NULL`.

→ *sortInfoIDP*
Pointer to new `sortInfoID` variable, or `NULL`.

→ *typeP*
Pointer to new `type` variable, or `NULL`.

→ *creatorP*
Pointer to new `creator` variable, or `NULL`.

Returns Returns `errNone` if no error or one of the following if an error occurred:

- `dmErrInvalidDatabaseName`
- `dmErrAlreadyExists`
- `dmErrInvalidParam`

Comments When this call changes `appInfoID` or `sortInfoID`, the old chunk ID (if any) is marked as an orphaned chunk[1] and the new chunk ID is unorphaned. Consequently, you shouldn't replace an existing `appInfoID` or `sortInfoID` if that chunk has already been attached to another database.

Call this routine to set any or all information about a database except for the card number and database ID. This routine sets the new value for any non-NULL parameter.

See Also DmDatabaseInfo(), DmOpenDatabaseInfo(), DmFindDatabase(), DmGetNextDatabaseByTypeCreator(), TimDateTimeToSeconds()

DmSetRecordInfo Function

Purpose Set record information stored in the database header.

Declared In DataMgr.h

Prototype

```
Err DmSetRecordInfo (DmOpenRef dbP, UInt16 index,
   UInt16 *attrP, UInt32 *uniqueIDP)
```

Parameters

→ *dbP*
DmOpenRef to open database.

→ *index*
Index of record.

→ *attrP*
Pointer to new attribute variable, or NULL. See "Record Attribute Constants" for a list of possible values.

→ *uniqueIDP*
Pointer to new unique ID variable, or NULL.

Returns Returns errNone if no error, or one of the following if an error occurred:

- dmErrReadOnly

---

1. An "orphaned chunk" is one that is allocated in the storage heap, but to which nothing refers. If the orphaned chunk is not put into a database as a record, an appInfo block, or the like, and if the application doesn't keep track of it—in a global variable, perhaps—it could get lost. If the application doesn't get around to freeing the chunk before it quits or crashes, or before the device is reset, that storage will be forever unusable: the user can't delete it since the user only deletes databases.

   During a soft reset, the OS walks through the storage heap and frees any orphaned chunks that it finds. Since most users reset only rarely, however, you shouldn't rely on this happening.

- dmErrNotRecordDB
- dmErrIndexOutOfRange

Some releases may display a fatal error message instead of returning the error code.

Comments Sets information about a record. This routine cannot be used to set the dmRecAttrBusy bit; instead, use DmGetRecord() to set the bit and DmReleaseRecord() to clear it.

Normally, the unique ID for a record is automatically created by the data manager when a record is created using DmNewRecord(), so an application would not typically change the unique ID.

Compatibility If Palm OS Cobalt Feature Set is present and the call to this function involves a PIM application database, PACE intercepts the call and takes the appropriate action on the corresponding schema database. For more information see "Accessing the PIM Application Databases" on page 2451.

See Also DmNumRecords(), DmRecordInfo()

DmSetResourceInfo Function

Purpose Set information on a given resource.

Declared In DataMgr.h

Prototype

```
Err DmSetResourceInfo (DmOpenRef dbP,
   UInt16 index, DmResType *resTypeP,
   DmResID *resIDP)
```

Parameters

→ *dbP*
DmOpenRef to open database.

→ *index*
Index of resource to set info for.

→ *resTypeP*
Pointer to new resType (resource type), or NULL.

→ *resIDP*
Pointer to new resource ID, or NULL.

Returns Returns errNone if no error, or one of the following if an error occurred:

- dmErrIndexOutOfRange
- dmErrReadOnly

May display a fatal error message if the database is not a resource database.

Comments Use this routine to set all or a portion of the information on a particular resource. Any or all of the new info pointers can be `NULL`. If not `NULL`, the type and ID of the resource are changed to `*resTypeP` and `*resIDP`.

DmStrCopy Function

Purpose Copies a string to a record within a database that is open for writing.

Declared In `DataMgr.h`

Prototype

```
Err DmStrCopy (void *recordP, UInt32 offset,
    const Char *srcP)
```

Parameters

↔ *recordP*
Pointer to data record (chunk pointer).

→ *offset*
Offset within record to start writing.

→ *srcP*
Pointer to null-terminated string.

Returns Returns `errNone` if no error. May display a fatal error message if the record pointer is invalid or the function overwrites the record.

Comments This is one of the routines that must be used to write to Data Manager records; because the data storage area is write-protected, you cannot write to it directly. This routine checks the validity of the chunk pointer for the record to insure that writing the record will not exceed the chunk bounds. DmStrCopy is a convenience method that determines the size of the supplied string and then simply calls DmWrite().

See Also DmSet()

DmSync Function

Purpose Update the automatic backup files for all open databases. This function is deprecated; developers should use `DmSyncDatabase()` instead.

Declared In `DataMgr.h`

Prototype

```
void DmSync(void)
```

Parameters None.

Returns Nothing.

Compatibility This function is not supported on Palm OS Cobalt and is deprecated on Palm OS Garnet. Developers should use `DmSyncDatabase()` instead.

DmSyncDatabase Function

Purpose Update the automatic backup file for a given open database.

Declared In `DataMgr.h`

Prototype

```
Err DmSyncDatabase (DmOpenRef dbRef)
```

Parameters

→ *dbRef*
: A `DmOpenRef` to an open database.

Returns Returns `errNone` if no error, or one of the following if an error occurs:

`dmErrInvalidParam`
: *dbRef* isn't a valid reference to an open database.

`dmErrReadOnly`
: *dbRef* references a database that is open in read-only mode. The database must be open for writing

`dmErrOperationAborted`
: The Palm OS device doesn't support the automatic database backup feature.

Comments The database is left open.

Use this function to cause an open database to be backed up.

Many devices running Palm OS Cobalt version 6.1 will back up the contents of the RAM storage heaps to some sort of non-volatile

NAND flash. In the event that the RAM storage heaps are corrupted or are lost for some reason, the storage heaps can then be restored to their saved state. Backup is automatically triggered on a limited set of events: database close, database create, a call to `DmSetDatabaseInfo()`, or upon device sleep (open databases only). Developers can explicitly cause a database to be backed up by calling `DmSyncDatabase()`.

For additional information on this feature, see "Automatic Database Backup and Restore" on page 204.

Compatibility In order to determine whether or not a given device has this capability, check for the presence of the `sysFtrNumDmAutoBackup` feature.

DmWrite Function

Purpose Copies a specified number of bytes to a record within a database that is open for writing.

Declared In `DataMgr.h`

Prototype

```
Err DmWrite (void *recordP, UInt32 offset,
   const void *srcP, UInt32 bytes)
```

Parameters

↔ *recordP*
Pointer to locked data record (chunk pointer).

→ *offset*
Offset within record to start writing.

→ *srcP*
Pointer to data to copy into record.

→ *bytes*
Number of bytes to write.

Returns Returns `errNone` if no error. May display a fatal error message if the record pointer is invalid or the function overwrites the record.

Comments This is one of the routines that must be used to write to Data Manager records; because the data storage area is write-protected, you cannot write to it directly. This routine checks the validity of the

chunk pointer for the record to insure that writing the record will not exceed the chunk bounds.

See Also DmStrCopy(), DmSet()

DmWriteCheck Function

Purpose Check the parameters of a write operation to a data storage chunk before actually performing the write.

Declared In DataMgr.h

Prototype

```
Err DmWriteCheck (void *recordP, UInt32 offset,
   UInt32 bytes)
```

Parameters

→ *recordP*
Locked pointer to recordH.

→ *offset*
Offset into record to start writing.

→ *bytes*
Number of bytes to write.

Returns Returns errNone if no error; returns dmErrNotValidRecord or dmErrWriteOutOfBounds if an error occurred.

Application-Defined Functions

DmComparF Function

Purpose Compares two records in a database.

Declared In DataMgr.h

Prototype

```
typedef Int16 DmComparF (void *rec1, void *rec2,
   Int16 other, SortRecordInfoPtr rec1SortInfo,
   SortRecordInfoPtr rec2SortInfo,
   MemHandle appInfoH)
```

Parameters

→ *rec1, rec2*
Pointers to the two records to compare.

→ *other*
Any other custom information you want passed to the comparison function. This parameter is often used to indicate a sort direction (ascending or descending).

→ *rec1SortInfo, rec2SortInfo*
Pointers to `SortRecordInfoType` structures that specify unique sorting information for the two records.

→ *appInfoH*
A handle to the database's application info block.

Returns

Returns:

- 0 if `rec1` = `rec2`.
- < 0 if `rec1` < `rec2`.
- > 0 if `rec1` > `rec2`.

Comments

This function is used to sort the records in a database. It is specifically called by `DmFindSortPosition()`, `DmInsertionSort()`, and `DmQuickSort()`.

Compatibility

The `DmComparF` prototype changed in Palm OS version 2.0. Previously, the prototype only defined the first three parameters.

As a rule, the change in the number of arguments from three to six doesn't cause problems when a 1.0 application is run on a 2.0 device because the system only pulls the arguments from the stack that are there.

Keep in mind, however, that some optimized applications built with tools other than Metrowerks CodeWarrior for Palm OS may have problems as a result of the change in arguments when running on a 2.0 or later device.

31

Debug Manager

The Debug Manager provides a set of debugger functions that go above and beyond what is provided in ROM.

Debug Manager Structures and Types 683
Debug Manager Constants 685
Debug Manager Functions and Macros 687
Application-Defined Functions 690

The header file `DebugMgr.h` declares the API that this chapter describes.

Debug Manager Structures and Types

DbgCtlEnumInfoType Struct

Purpose

Declared In `DebugMgr.h`

Prototype

```
typedef struct DbgCtlEnumInfoType {
   DbgCtlEnumCallbackFunc *enumFuncP;
   void *callbackDataP;
} DbgCtlEnumInfoType
```

Fields

`enumFuncP`

`callbackDataP`

Comments

DbgCtlHandlerInfoType Struct

Purpose

Declared In `DebugMgr.h`

Prototype

```
typedef struct DbgCtlHandlerInfoType {
   DbgControlFuncType *handlerFuncP;
   UInt32 version;
   Boolean enabled;
   Char name[dbgCtlHandlerNameLen+1];
   Char ver[dbgCtlHandlerVerLen+1];
   UInt8 reserved1;
   UInt32 dwReserved;
} DbgCtlHandlerInfoType
```

Fields

`handlerFuncP`
: Pointer to the handler's `DbgControl()` function. This function's prototype must match that declared by `DbgControlFuncType()`.

`version`
: Numeric version number (handler-defined).

`enabled`
: `true` if the handler is enabled; `false` if it isn't.

`name`
: Null-terminated handler name. This name should not exceed `dbgCtlHandlerNameLen` characters in length (not counting the null terminator).

`ver`
: Null-terminated handler version string. This string should not exceed `dbgCtlHandlerVerLen` characters in length (not counting the null terminator).

`reserved1`
: Reserved for future use.

`dwReserved`
: Reserved field. *The caller must initialize this field to zero*.

Debug Manager Constants

DbgControl() Constants

Purpose Constants used in conjunction with `DbgControl()`.

Declared In `DebugMgr.h`

Constants `extern DbgControlFuncType DbgControl`

`#define dbgCtlAllHandlersID 0`
Value that can be supplied to `DbgControlFuncType()` to indicate that the operation is to be handled by all handlers.

`#define dbgCtlFirstCustomOp 0x8000`
Debug handler operations are numbered beginning with this value. The system reserves all operation number values less than this one.

`#define dbgCtlHandled true`
Value returned by `DbgControlFuncType()` to indicate that the debug control operation was handled.

`#define dbgCtlHandlerNameLen 31`
Maximum handler name length, not including the terminating null character.

`#define dbgCtlHandlerVerLen 15`
Maximum handler version string length, not including the terminating null character.

`#define dbgCtlNotHandled false`
Value returned by `DbgControlFuncType()` to indicate that the debug control operation was not handled.

System-Defined Debug Control Operations Enum

Purpose Opcodes for system-defined debug control operations. These values are passed in the *op* parameter of the `DbgControlFuncType()` function.

Declared In `DebugMgr.h`

Constants

`dbgCtlOpEnumHandlers = 1`
: Enumerate all handlers. The *handlerID* parameter must be set to `dbgCtlAllHandlersID`.

`dbgCtlOpGetHandlerInfo`
: Get information about a specific handler.

`dbgCtlOpEnableHandler`
: Enable the specified handler. `dbgCtlAllHandlersID` can be supplied for the *handlerID* parameter to enable all handlers. The *paramP* and *dwParamP* parameters are ignored.

`dbgCtlOpDisableHandler`
: Disable the specified handler. `dbgCtlAllHandlersID` can be supplied for the *handlerID* parameter to disable all handlers. The *paramP* and *dwParamP* parameters are ignored.

`dbgCtlOpGetEnabledStatus`
: Determine whether a specific handler is enabled. When making this call, *dwParamP* should point to a `UInt32` that will be set to a non-zero value if the handler is enabled, or to zero if it is disabled. The *paramP* parameter is ignored.

`dbgCtlOpGetVersion`
: Obtain the version number for a specific handler. When making this call, *dwParamP* should point to a `UInt32` that will receive the handler's version number. The *paramP* parameter is ignored.

`dbgCtlOpLAST`
: Not a valid operation code, this value represents the next unused operation code value.

Debug Manager Functions and Macros

DbgBreak Function

Purpose Connects to the assembly-level debugger and halts execution.

Declared In `DebugMgr.h`

Prototype

```
void DbgBreak (void)
```

Parameters None.

Returns Nothing.

See Also `DbgSrcBreak()`

DbgCommSettings Function

Purpose Retrieve or change the communications settings used by the debugger.

Declared In `DebugMgr.h`

Prototype

```
Err DbgCommSettings (UInt32 *baudP,
    UInt32 *flagsP)
```

Parameters

↔ *baudP*

To retrieve the current settings, set **baudP* to zero. Upon return, **baudP* will contain the current baud rate. To change the current settings, set **baudP* to the new baud rate.

↔ *flagsP*

If **baudP* is set to zero, the communications settings flags are retrieved along with the baud rate and stored in **flagsP*. If **baudP* is nonzero, **flagsP* should contain the desired communications settings flags; see "Serial Settings Constants" on page 1601 for the set of flags that can be combined to make up **flagsP*.

Returns `errNone` if the operation completed successfully, or `serErrNotOpen` if the debugger communications library is not open.

DbgGetMessage Function

Purpose Wait for, and then retrieve, a message packet from the host.

Declared In `DebugMgr.h`

Prototype

```
Char *DbgGetMessage (UInt8 *bufferP,
    Int32 timeout)
```

Parameters

← *bufferP*
Pointer to a pre-allocated buffer into which the message is to be written. When allocating memory for this buffer , note that it must be at least `sysPktMaxBodySize` bytes in length.

→ *timeout*
Maximum number of system ticks to wait for the beginning of the packet. A timeout value of `-1` means that the function should wait forever.

Returns A pointer to the start of the text in the received packet, or zero if an error occurred. Note that the returned pointer will *not* be the same as the supplied buffer pointer.

Comments Use this function to obtain text input from the serial port.

See Also `DbgMessage()`

DbgInit Function

Purpose Initialize the debugger and the debugger's serial port.

Declared In `DebugMgr.h`

Prototype

```
Int32 DbgInit (MemPtr spaceP,
    MemPtr dispatchTableP[], Boolean openComm)
```

Parameters

← *spaceP*
Pointer to a pre-allocated block of memory to be used for debugger globals. This block of memory should be at least 64 bytes in length.

→ *dispatchTableP*
Pointer to system dispatch table, where debugger traps get installed.

→ *openComm*
Pass `true` to open communications and display a welcome message, or `false` to skip this step.

Returns The number of bytes used for globals.

Comments This function is called for you at boot time. Applications do not need to call this function.

DbgMessage Function

Purpose Sends a null-terminated string to the current debug device.

Declared In `DebugMgr.h`

Prototype
```
void DbgMessage (const Char *aStr)
```

Parameters → *aStr*
The null-terminated string to be sent to the current debug device.

Returns Nothing.

See Also DbgGetMessage(), DbgSrcMessage()

DbgSrcBreak Function

Purpose Connects to the source-level debugger and halts execution.

Declared In `DebugMgr.h`

Prototype
```
void DbgSrcBreak (void)
```

Parameters None.

Returns Nothing.

See Also DbgBreak()

DbgSrcMessage Function

Purpose Displays a message to the source-level debugger.

Declared In `DebugMgr.h`

Prototype
```
void DbgSrcMessage (const Char *debugStr)
```

Parameters → *debugStr*
A null-terminated string.

Returns Nothing.

See Also DbgMessage()

Application-Defined Functions

DbgControlFuncType Function

Purpose Debug control function for implementing debug tracing and other such functions via debug handlers. The system's default implementation of this function (named `DbgControl()`) does nothing, simply returning `dbgCtlNotHandled`.

Declared In `DebugMgr.h`

Prototype

```
Boolean DbgControlFuncType (UInt32 handlerID,
    UInt16 op, void *paramP, UInt32 *dwParamP)
```

Parameters

→ *handlerID*
ID of the debug handler that is to handle this call. A value of `dbgCtlAllHandlersID` indicates that each handler is to execute the specified operation and then pass the call down the chain for the next handler to execute.

→ *op*
Operation ID specific to the handler specified by *handlerID*. See "System-Defined Debug Control Operations" on page 686 for a number of operation IDs defined by the system.

↔ *paramP*
pointer to data specific to the reqested operation.

↔ *dwParamP*
Pointer to a `UInt32` value specific to the requested operation.

Returns `dbgCtlHandled` if the specified handler handled the requested operation, or `dbgCtlNotHandled` if it didn't.

Comments The default implementation does nothing, leaving the real work up to "debug handlers." Debug handlers will be implemented as system extensions. As they are loaded, extensions will override the

`DbgControl()` function call and chain to those handlers loaded before them.

When a debug control call is made by the client, a handler ID of the handler that implements the functionality is passed in as the first parameter. The handler ID is the unique creator ID of the handler. When a handler is called, it will first examine the handler ID: if it matches its own, the handler will execute the command and return; if the handler ID does not match, the handler must pass the call down the chain and return the value from that call.

The operation to be performed is indicated by the parameter *op*. Operation codes are specific to each handler, meaning that that the same operation code values may be used by different handlers. Operation code values defined by handlers must begin at `dbgCtlFirstCustomOp`. The special handler ID of `dbgCtlAllHandlersID` applies to all handlers. When this handler ID is passed, each handler is responsible for executing the requested action and then passing the call down the chain.

The last two parameters are defined by each handler for its own operations.

DbgCtlEnumCallbackFunc Function

Purpose

Declared In `DebugMgr.h`

Prototype

```
void DbgCtlEnumCallbackFunc (void *callbackDataP,
    UInt32 handlerID,
    DbgControlFuncType *handlerFuncP)
```

Parameters

`callbackDataP`

→ *`handlerID`*

`handlerFuncP`

Returns Nothing.

32

Error Manager

This chapter provides reference material for the error manager. The error manager API is declared in the header files `ErrorMgr.h` and `ErrorBase.h`. This chapter covers:

ERROR_CHECK_LEVEL Define 693

Error Manager Data Structures 694

Error Manager Functions and Macros 694

For more information on the error manager, see the chapter "Debugging Strategies" in the *Palm OS Programmer's Companion*, vol. I.

ERROR_CHECK_LEVEL Define

The error manager uses the compiler define `ERROR_CHECK_LEVEL` to control the level of error messages displayed. You can set the value of the compiler define to control which level of error checking and display is compiled into the application. Three levels of error checking are supported: none, partial, and full.

If you set ERR_CHECK_LEVEL to...	The compiler...
`ERROR_CHECK_NONE (0)`	Doesn't compile in any error calls.
`ERROR_CHECK_PARTIAL (1)`	Compiles in only `ErrDisplay` and `ErrFatalDisplayIf` calls.
`ERROR_CHECK_FULL (2)`	Compiles in all three calls.

Error Manager Data Structures

ErrExceptionType Struct

Purpose An `ErrExceptionType` structure is created for each `ErrTry()` and `ErrCatch()` block. At any point in the program, there is a linked list of these structures. An `ErrExceptionType` structure stores information about the state of the machine (register values) at the start of the `ErrTry` block.

Prototype

```
typedef struct ErrExceptionType {
  struct ErrExceptionType *nextP;
  ErrJumpBuf state;
  Int32 err;
} ErrExceptionType;
typedef ErrExceptionType *ErrExceptionPtr;
```

Fields

`nextP`
: Next `ErrExceptionType` structure in the linked list.

`state`
: Storage for setjmp/longjmp.

`err`
: Error code.

Error Manager Functions and Macros

ErrAlert Macro

Purpose Displays an alert dialog for runtime errors.

Declared In `ErrorBase.h`

Prototype

```
#define ErrAlert (err)
```

Parameters

→ *err*
: An error code (as type `Err`).

Returns Returns 0, which indicates that the OK button has been clicked to dismiss the dialog.

Comments This macro is intended for use by applications that are likely to receive runtime errors when the application itself is not at fault. For example, a networking application might use it to display an alert if the remote server cannot be found.

The error message displayed on the dialog is stored in a 'tSTL' resource. A 'tSTL' resource contains a list of strings that can be looked up by index. The `err` parameter is used as the index into this list.

To use application-defined error codes in `ErrAlert`, make sure that all of your error codes are greater than or equal to `appErrorClass`. This way, the error manager looks up the code in the application's 'tSTL' resource number 0. All other error codes are taken from 'tSTL' resource stored in the system.

Compatibility Implemented only if 3.2 New Feature Set is present.

ErrCatch Macro

Purpose Marks the end of an `ErrTry()` block and the beginning of an `ErrCatch` block.

Declared In `ErrorBase.h`

Prototype

```
#define ErrCatch (theErr)
```

Parameters → *`theErr`*

An exception code identifying the reason for the failure. This is the value supplied to the `ErrThrow()` call that caused the jump to this `ErrCatch` block.

Returns Returns nothing.

Comments `ErrCatch` can only be used in conjunction with `ErrTry()` and `ErrEndCatch`. See the comments under `ErrTry` for usage instructions.

`ErrTry`, `ErrCatch` and `ErrThrow` are based on setjmp and longjmp. At the beginning of an `ErrTry` block, setjmp saves the machine registers. `ErrThrow` calls longjmp, which restores the registers and jumps to the beginning of the `ErrCatch` block. Therefore, any changes in the `ErrTry` block to variables stored in registers aren't retained when entering the `ErrCatch` block.

The solution is to declare variables that you want to use in both the `ErrTry` and `ErrCatch` blocks as "volatile". For example:

```
volatile long x = 1;  // Declare volatile local variable
ErrTry {
   x = 100;    // Set local variable in Try
   ErrThrow(-1);
}
ErrCatch(inErr) {
   if (x > 1) {    // Use local variable in Catch
      SysBeep(1);
   }
} ErrEndCatch
```

If you have many local variables after the `ErrCatch` you may want to put the `ErrTry` and `ErrCatch` in a separate enclosing function.

ErrDisplay Macro

Purpose Displays an error alert if error checking is set to partial or full.

Declared In `ErrorMgr.h`

Prototype

```
#define ErrDisplay (msg)
```

Parameters → *msg*

Error message text as a string.

Returns No return value.

Comments Call this macro to display an error message, source code filename, and line number. This macro is compiled into the code only if the compiler define `ERROR_CHECK_LEVEL` is set to 1 or 2 (`ERROR_CHECK_PARTIAL` or `ERROR_CHECK_FULL`).

See Also `ErrFatalDisplayIf()`, `ErrNonFatalDisplayIf()`

ErrDisplayFileLineMsg Function

Purpose Display a dialog with an error message. Do not allow the user to exit the dialog or continue.

Declared In `ErrorBase.h`

Prototype

```
void ErrDisplayFileLineMsg
    (const Char *const filename, UInt16 lineNo,
    const Char *const msg)
```

Parameters

→ *filename*
Source code filename.

→ *lineno*
Line number in the source code file.

→ *msg*
Message to display.

Returns Never returns.

Comments Called by `ErrFatalDisplayIf()` and `ErrNonFatalDisplayIf()`. This function is useful when the application is already on the device and being tested by users.

On Japanese systems, the system displays a generic message indicating that an error has occurred instead of displaying the English message.

See Also `ErrFatalDisplayIf()`, `ErrNonFatalDisplayIf()`, `ErrDisplay()`

ErrEndCatch Macro

Purpose Marks the end of an `ErrCatch()` block.

Declared In `ErrorBase.h`

Prototype

```
#define ErrEndCatch
```

Parameters `None.`

Returns Returns nothing.

Comments `ErrEndCatch` can only be used in conjunction with `ErrTry()` and `ErrCatch`. See the comments under `ErrTry` for usage instructions.

ErrExceptionList Function

Purpose Return the address of the pointer to the first `ErrExceptionType` structure in the linked list of exception structures.

Declared In `ErrorBase.h`

Prototype
```
MemPtr *ErrExceptionList (void)
```

Parameters `None.`

Returns Returns the address of the pointer to the first `ErrExceptionType` structure linked into the exception list.

Comments This function is used by the ErrTry and ErrCatch macros as well as the `ErrThrow()` function in order to find the exception list.

When called from an application, this routine returns the application's exception list.

ErrFatalDisplayIf Macro

Purpose Displays an error alert dialog if `condition` is `true` and error checking is set to partial or full.

Declared In `ErrorMgr.h`

Prototype
```
#define ErrFatalDisplayIf (condition, msg)
```

Parameters

→ *condition*
: A boolean value. If `true`, display the error.

→ *msg*
: Error message text as a string.

Returns No return value.

Comments Call this macro to display a fatal error message, source code filename, and line number. The alert is displayed only if `condition` is `true`. The dialog is cleared only when the user resets the system by responding to the dialog.

This macro is compiled into the code if the compiler define `ERROR_CHECK_LEVEL` is set to 1 or 2 (`ERROR_CHECK_PARTIAL` or `ERROR_CHECK_FULL`).

See Also `ErrNonFatalDisplayIf()`, `ErrDisplay()`,

ErrNonFatalDisplayIf Macro

Purpose Displays an error alert dialog if condition is true and error checking is set to full.

Declared In `ErrorMgr.h`

Prototype

```
#define ErrNonFatalDisplayIf (condition, msg)
```

Parameters

→ *condition*
A boolean value. If `true`, display the error.

→ *msg*
Error message text as a string.

Returns No return value.

Comments Call this macro to display a nonfatal error message, source code filename, and line number. The alert is displayed only if `condition` is `true`. The alert dialog is cleared when the user selects to continue (or resets the system).

This macro is compiled into the code only if the compiler define `ERROR_CHECK_LEVEL` is set to 2 (`ERROR_CHECK_FULL`).

See Also ErrFatalDisplayIf(), ErrDisplay()

ErrThrow Function

Purpose Cause a jump to the nearest Catch block.

Declared In `ErrorBase.h`

Prototype

```
void ErrThrow (Int32 err)
```

Parameters

→ *err*
Error code.

Returns Never returns.

Comments Use the macros ErrTry, ErrCatch, and ErrEndCatch in conjunction with this function.

See Also `ErrFatalDisplayIf()`, `ErrNonFatalDisplayIf()`, `ErrDisplay()`

ErrTry Macro

Purpose Marks the beginning of a try/catch block.

Declared In `ErrorBase.h`

Prototype `#define ErrTry`

Parameters `None.`

Returns Returns nothing.

Comments An exception raised by a call to `ErrThrow()`—even from within a nested subroutine—causes program execution to switch to the beginning of the `ErrCatch()` block. If the end of the block enclosed by `ErrTry` is encountered without a call to `ErrThrow`, execution jumps to the line of code following the `ErrEndCatch()` macro. See "The Try-and-Catch Mechanism" on page 424 of the *Palm OS Programmer's Companion*, vol. I for a more thorough description of how this mechanism works.

Example You must structure your code exactly as shown here. You can't use `ErrTry` without `ErrCatch` and `ErrEndCatch`, or vice versa.

```
ErrTry {
   // Do something which may fail. Call ErrThrow to signal
   // failure and force a jump to the following ErrCatch
   // block.
}
ErrCatch(inErr) {
   // Recover or cleanup after a failure in the above ErrTry
   // block. "inErr" is an exception code identifying the
   // reason for the failure.

   // Call ErrThrow if you want to jump out to the next
   // ErrCatch block.

   // The code in this block doen't execute if the above
   // ErrTry block completes without a call to ErrThrow.
} ErrEndCatch
```

33

Expansion Manager

This chapter provides the following information about the Expansion Manager:

- Expansion Manager Data Structures
- Expansion Manager Constants
- Expansion Manager Functions

The header file `ExpansionMgr.h` declares the Expansion Manager API. For more information on the Expansion Manager, see Chapter 7, "Expansion," in *Palm OS Programmer's Companion*, vol. I.

Note that the Expansion Manager is an optional system extension; the functions described in this chapter are implemented only if the Expansion Manager Feature Set is present.

Expansion Manager Data Structures

ExpCardInfoType Struct

Purpose

The `ExpCardInfoType` declaration defines a structure that is passed to `ExpCardInfo()`. This structure is used to determine the characteristics of the card loaded in the slot. It is initialized by the underlying slot driver with the following information.

Prototype

```
typedef struct ExpCardInfoTag {
  UInt32 capabilityFlags;
  Char
    manufacturerStr[expCardInfoStringMaxLen+1];
  Char productStr[expCardInfoStringMaxLen+1];
  Char
    deviceClassStr[expCardInfoStringMaxLen+1];
  Char
  deviceUniqueIDStr[expCardInfoStringMaxLen+1];
} ExpCardInfoType, *ExpCardInfoPtr;
```

Fields `capabilityFlags`
Describes the capabilities of the card. The following flags are currently supported:

`expCapabilityHasStorage`
Indicates that the card supports reading and (possibly) writing.

`expCapabilityHidden`
Indicates that the expansion card should not be visible to the user.

`expCapabilityNonRemovable`
Indicates that the expansion card cannot be removed.

`expCapabilityReadOnly`
Indicates that the card is read only.

`expCapabilitySerial`
Indicates that the card supports a simple serial interface.

`manufacturerStr`
Names the manufacturer of the card. For example "Palm" or "Motorola".

`productStr`
Name of the product. For example "SafeBackup 32MB".

`deviceClassStr`
Describes the type of card, for example, "Backup" or "Ethernet".

`deviceUniqueIDStr`
Unique identifier for the product. A serial number for example. This value is set to the empty string if no identifier exists.

Expansion Manager Constants

Error Codes

The Expansion Manager defines the following error codes:

Constant	Description
`expErrUnsupportedOperation`	The operation is unsupported or undefined.
`expErrNotEnoughPower`	The required power is not available.
`expErrCardNotPresent`	There is no card present in the given slot.
`expErrInvalidSlotRefNum`	The slot reference number is not valid.
`expErrSlotDeallocated`	The slot reference number is within the valid range, but the slot has been deallocated.
`expErrCardNoSectorReadWrite`	The card does not support the slot driver block read/write API.
`expErrCardReadOnly`	The card supports the slot driver block read/write API but the card is read only.
`expErrCardBadSector`	The card supports the slot driver block read/write API but the sector is bad.
`expErrCardProtectedSector`	The card supports the slot driver block read/write API but the sector is protected.
`expErrNotOpen`	The slot driver library has not been opened.
`expErrStillOpen`	The slot driver library is still open; it may have been opened more than once.
`expErrUnimplemented`	The call is unimplemented.
`expErrEnumerationEmpty`	There are no values remaining to enumerate.
`expErrIncompatibleAPIVer`	The API version of the underlying slot driver is not supported by this version of Expansion Manager.

Defined Media Types

The following media types are defined by the Expansion Manager. These media types are used with the function `VFSVolumeInfo()` in the `VolumeInfoType.mediaType` field. The media type is also passed as a parameter to the

VFSRegisterDefaultDirectory() and VFSUnregisterDefaultDirectory() functions.

Constant	Value	Description
expMediaType_Any	'wild'	Matches all media types when looking up a default directory
expMediaType_MemoryStick	'mstk'	Memory stick
expMediaType_CompactFlash	'cfsh'	Compact Flash
expMediaType_SecureDigital	'sdig'	Secure Digital
expMediaType_MultiMediaCard	'mmcd'	MultiMedia Card
expMediaType_SmartMedia	'smed'	SmartMedia
expMediaType_RAMDisk	'ramd'	A RAM disk based media
expMediaType_PoserHost	'pose'	Host file system emulated by the Palm OS® Emulator
expMediaType_MacSim	'PSim'	Host file system emulated by the Mac Simulator

Expansion Manager Functions

ExpCardGetSerialPort Function

Purpose Get a card's serial port creator ID for use in serial access.

Declared In `ExpansionMgr.h`

Prototype

```
Err ExpCardGetSerialPort(UInt16 slotRefNum,
   UInt32 *portP)
```

Parameters

→ *slotRefNum*
Slot number of slot to check.

← *portP*
Pointer to `UInt32` into which the serial port creator ID is stored.

Returns Returns the following result codes:

`errNone`
No error

`expErrInvalidSlotRefNum`
The specified slot number is invalid.

`expErrSlotDeallocated`
The specified slot number is within the valid range but has been deallocated.

Compatibility Implemented only if the Expansion Manager Feature Set is present. If Palm OS Cobalt Feature Set is present this function is unimplemented.

See Also `ExpCardInfo()`

ExpCardInfo Function

Purpose Obtains information about a card in a given slot.

Declared In `ExpansionMgr.h`

Prototype

```
Err ExpCardInfo(UInt16 slotRefNum,
   ExpCardInfoType *infoP)
```

Parameters

→ *slotRefNum*
Slot number.

← *infoP*
Pointer to `ExpCardInfoType` structure.

Returns Returns one of the following result codes:

`errNone`
No error

`expErrCardNotPresent`
There is no card present in the specified slot.

`expErrInvalidSlotRefNum`
The slot number is invalid.

`expErrSlotDeallocated`
The slot number is within the valid range but has been deallocated.

Comments This function returns information about a card, including whether the card supports secondary storage or is strictly read-only, by filling in the `ExpCardInfoType` structure's fields.

Compatibility Implemented only if the Expansion Manager Feature Set is present.

See Also `ExpCardGetSerialPort()`, `ExpCardPresent()`, `ExpSlotEnumerate()`

ExpCardPresent Function

Purpose Determines if a card is present in the given slot.

Declared In `ExpansionMgr.h`

Prototype

```
Err ExpCardPresent(UInt16 slotRefNum)
```

Parameters

→ *slotRefNum*
Slot number.

Returns Returns the following result codes:

`errNone`
A card is present in the specified slot.

`expErrCardNotPresent`
There is no card present in the specified slot.

`expErrInvalidSlotRefNum`
The specified slot number is not valid.

`expErrSlotDeallocated`
The specified slot number is within the valid range but has been deallocated.

Comments The Expansion Manager passes the call through to the appropriate slot driver.

Compatibility Implemented only if the Expansion Manager Feature Set is present.

See Also `ExpCardInfo()`, `ExpSlotEnumerate()`

ExpSlotDriverInstall Function

Purpose Installs and initializes a slot driver shared library into the system table.

Declared In `ExpansionMgr.h`

Prototype

```
Err ExpSlotDriverInstall (UInt32 dbCreator,
   UInt16 *slotLibRefNumP)
```

Parameters

→ *`dbCreator`*
Database creator code of the slot driver library to be installed.

→ *`slotLibRefNumP`*
Pointer to variable for returning the library reference number (on failure, `sysInvalidRefNum` is returned in this variable).

Returns Returns `errNone` if the slot driver is installed correctly. Because this function uses `SysLibInstall()` to install the slot driver shared library, `ExpSlotDriverInstall` may return any of the error codes that `SysLibInstall` returns, including `sysErrLibNotFound`, `sysErrNoFreeRAM`, and `sysErrNoFreeLibSlots`. It may also return any error code returned by `SlotOpen`, the implementation of which is specific to a given device manufacturer.

Comments This function is not typically called by applications but can be used to load additional slot drivers after the device has booted. It is called internally by the Expansion Manager to install a slot driver shared library into the library table and initialize it for use. Once installed, the `slotLibRefNum` can be used by other functions to refer to the library.

Compatibility Implemented only if the Expansion Manager Feature Set is present. If Palm OS Cobalt Feature Set is present this function is unimplemented.

See Also `ExpSlotDriverRemove()`, `ExpSlotLibFind()`

ExpSlotDriverRemove Function

Purpose Closes and remove a slot driver shared library from the system table.

Declared In `ExpansionMgr.h`

Prototype

```
Err ExpSlotDriverRemove (UInt16 slotLibRefNum)
```

Parameters → *`slotLibRefNum`*
Slot driver shared library reference number.

Returns Returns `errNone`.

Comments This function is not typically called by applications but can be used to unload slot drivers associated with external slots. It is called internally by the Expansion Manager to remove the shared library from the system table, and, if appropriate, release the slot allocated to the given `slotLibRefNum`. Prior to removing the slot driver, it unmounts any volumes associated with the slot.

Compatibility Implemented only if the Expansion Manager Feature Set is present. If Palm OS Cobalt Feature Set is present this function is unimplemented.

See Also ExpSlotDriverInstall(), ExpSlotLibFind(), SysLibRemove()

ExpSlotEnumerate Function

Purpose Iterates through valid slot numbers.

Declared In `ExpansionMgr.h`

Prototype

```
Err ExpSlotEnumerate(UInt16 *slotRefNumP,
   UInt32 *slotIteratorP)
```

Parameters

← *slotRefNumP*
: Reference number of the currently-enumerated slot.

→ *slotIteratorP*
: Pointer to the index of the last entry enumerated. For the first iteration, initialize this parameter to the constant `expIteratorStart`. Upon return this references the next entry in the directory. If this is the last entry, this parameter is set to `expIteratorStop`.

Returns Returns one of the following result codes:

`errNone`
: The slot reference number indicated by `slotRefNumP` is valid.

`expErrEnumerationEmpty`
: There are no slots left to enumerate. `slotRefNumP` is set to the slot number of the currently enumerated slot or `invalidSlotRefNum` if there are no slots.

Comments This function iterates through the device's slots. The first time this function is called, set `*slotIteratorP` to `expIteratorStart` to find the initial slot. Once set this value is changed with each subsequent call to this function until it reaches the maximum number of slots, at which point `ExpSlotEnumerate` sets `*slotIteratorP` to `expIteratorStop`.

Example The following is an example of how `ExpSlotEnumerate` should be used:

```
UInt16 slotRefNum;
UInt32 slotIterator = expIteratorStart;
while (slotIterator != expIteratorStop) {
   if ((err = ExpSlotEnumerate(&slotRefNum,
            &slotIterator)) == errNone) {
      // do something with the slotRefNum
   else {
      // handle error (by breaking out of the
```

```
        // loop, most likely
    }
}
```

Compatibility Implemented only if the Expansion Manager Feature Set is present.

See Also `ExpCardInfo()`, `ExpCardPresent()`, `ExpSlotDriverRemove()`

ExpSlotLibFind Function

Purpose Retrieves the slot driver library reference number for the driver that controls the specified slot.

Declared In `ExpansionMgr.h`

Prototype

```
Err ExpSlotLibFind(UInt16 slotRefNum,
    UInt16 *slotLibRefNum)
```

Parameters

→ *slotRefNum*
Slot number.

← *slotLibRefNum*
Pointer to the reference number for the slot driver library allocated to the given slot.

Returns Returns the following result codes:

`errNone`
No error.

`expErrInvalidSlotRefNum`
The slot number is invalid.

`expErrSlotDeallocated`
The slot number is within the valid range but has been deallocated.

Comments This function returns the reference number to the slot driver library loaded for the given `slotRefNum`. This function is used when making calls directly to the slot driver library.

Compatibility Implemented only if the Expansion Manager Feature Set is present. If Palm OS Cobalt Feature Set is present this function is unimplemented.

See Also `ExpSlotDriverInstall()`, `ExpSlotDriverRemove()`

34

Feature Manager

This chapter provides reference material for the feature manager. The feature manager API is declared in the header file `FeatureMgr.h`.

For more information on the feature manager, see the section "Features" in the *Palm OS Programmer's Companion*, vol. I.

To learn how to use the predefined Palm OS® features to test for the existence of certain OS features, see the "Compatibility Guide" appendix.

Feature Constants

This section lists the feature constants that various parts of Palm OS define to identify how the system works. You can obtain the values of these features with the `FtrGet()` function:

```
err = FtrGet(creator, constant, &value)
```

where *creator* is the constant listed in the Creator columns of the tables that follow, *constant* is the value listed in the Constant column, and *value* is the set by the `FtrGet` function if the feature is defined. If the feature is not defined, *value* is not set, and `sysErrNoSuchFeature` is returned. Unless otherwise specified, you should consider these features to be read-only.

The feature constants are grouped into the following categories:

- Networking Features
- Expansion and VFS Manager Features
- Hardware Capability Features
- Hardware Features
- Emulator and Debugging Features
- Localization Features
- Miscellaneous Features

Networking Features

Table 34.1 lists the features that provide information about the communications and networking libraries installed on the device.

Table 34.1 Communications feature constants

Constant	Creator	Value
`btexgFtrNumVersion`	`btexgFtrCreator`	The version number of the Bluetooth Exchange Library. See "Detecting the Bluetooth Exchange Library" on page 154 in the *Palm OS Programmer's Companion*, vol. II, *Communications* for more information.
`btLibFeatureVersion`	`btLibFeature Creator`	The version number of the Bluetooth Library. Used to determine if the Bluetooth Library Feature Set is present.
`inetFtrNumVersion`	`inetLibFtrCreato r`	The version number of the Internet Library, if present. See "System Requirements" on page 192 in *Palm OS Programmer's Companion*, vol. II, *Communications* for more information.
`irFtrNumVersion`	`irFtrCreator`	The version number of the IR library, if present.
`kCncFtrCncMgr Version`	`kCncFtrCncMgr Creator`	The version of the Connection Manager API. More than one version of this API has been released. If the value for this feature is greater than `0x00040001`, the Connection Manager Feature Set is present.
`kSmsFtrNumVersion`	`sysFileCSmsLib`	The version number of the SMS Exchange Library.

Table 34.1 Communications feature constants *(continued)*

Constant	Creator	Value
`sysFtrNumDefaultCompression`	`sysFtrCreator`	The default compression algorithm used for wireless networking. The Palm Web Clipping Application checks this feature. Note that the Palm Web Clipping Application is obsolete.
`netFtrCommandBlocks`	`netFtrCreator`	System use only.
`netFtrNumVersion`	`netFtrCreator`	The version number of the Net Library. See "Version Checking" on page 175 in the *Palm OS Programmer's Companion*, vol. II, *Communications* for more information.
`sysFtrBtSupportsScheduledWakeup`		The presence of this feature indicates that the Bluetooth hardware supports wakeup scheduling.
`sysFtrNewSerialPresent`	sysFileCSerialMgr	A nonzero value indicates that the New Serial Manager Feature Set is present.

Table 34.1 Communications feature constants *(continued)*

Constant	Creator	Value
`sysFtrNewSerialVersion`	sysFileCSerialMgr	The version of the New Serial Manager API. There is currently more than one version. See "Checking the Serial Manager Version" on page 94 in the *Palm OS Programmer's Companion*, vol. II, *Communications* for more information.
`sysFtrNumEncryption`	`sysFtrCreator`	One or more flags indicating which encryption schemes are present. The only currently supported encryption is DES, which can be tested for using `sysFtrNumEncryptionMaskDES`.

Expansion and VFS Manager Features

Table 34.2 lists the feature constants that provide information about the Expansion and VFS Managers.

Table 34.2 Expansion and VFS feature constants

Constant	Creator	Value
`expFtrIDVersion`	`sysFileCExpansionMgr`	The version number of the Expansion Manager if present on the device. Used to determine if the Expansion Manager Feature Set is present.
`vfsFtrIDDefaultFS`	`sysFileCVFSMgr`	The creator ID of the default file system that `VFSVolumeFormat()` uses when formatting new volumes. Application developers are allowed to set this value.
`vfsFtrIDVersion`	`sysFileCVFSMgr`	The version number of the VFS Manager. Use to determine if the VFS Manager Feature Set is present.

Hardware Capability Features

Table 34.3 lists the features constants that provide information about the capabilities of the hardware on which Palm OS is running.

Table 34.3 Hardware capability feature constants

Constant	Creator	Value
`kAttnFtr Capabilities`	`kAttnFtrCreator`	One or more of the Attention Manager Feature Constants, indicating the hardware's abilities for catching the user's attention (LED, sound, vibration, and so on). For more information, see "Detecting Device Capabilities" on page 326 in the *Palm OS Programmer's Companion*, vol. I.
`sysFtrNumBacklight`	`sysFtrCreator`	A nonzero value indicates that the device has a backlight.
`sysFtrNumDisplay Depth`	`sysFtrCreator`	The device's default display depth. Supported depths are 1, 2, 4, 8, and 16 bits per pixel.
`sysFtrNumDmAutoBackup`	`sysFtrCreator`	Indicates that the device has the Automatic Database Backup and Restore feature, which preserves device data even without a backup battery.
`sysFtrNumInputArea Flags`	`sysFtrCreator`	One or more of the flags listed in Table 34.4 on page 719 that provide information about the input area.
`sysFtrNumUIHardware Flags`		One or more flags specifying the set of UI navigation capabilities. See Table 34.5 on page 719. This feature is not currently used but may be used by future devices.

Table 34.4 describes the possible flags that could be set in the value for the `sysFtrNumInputAreaFlags` feature constant. You can AND these flags with the value returned by `FtrGet` to see if that attribute exists. For example:

```
err = FtrGet(sysFtrCreator, sysFtrNumInputAreaFlags,
  &flags);
if (!err) {
  if (flags & grfFtrInputAreaFlagDynamic) ...
}
```

Table 34.4 Input area attributes

Constant	Description
`grfFtrInputAreaFlagDynamic`	The input area is implemented in software. If 0, the input area is implemented as hardware.
`grfFtrInputAreaFlagCollapsible`	The user is allowed to collapse the input area, giving more room to the application area.
`grfFtrInputAreaFlagLiveInk`	The input area has the live ink feature, which echoes the pen stroke so that the user can see what has been written.

Table 34.5 describes the flags that may be set in the value for the `sysFtrNumUIHardwareFlags` feature constant.

Table 34.5 Navigation attributes

Constant	Description
`sysFtrNumUIHardwareHas5Way`	The device has a 5-way navigation rocker.
`sysFtrNumUIHardwareHasJog`	The device has a thumb wheel on the side.
`sysFtrNumUIHardwareHasJogBack`	The device has a thumb wheel with a back button.

Table 34.5 Navigation attributes

Constant	Description
`sysFtrNumUIHardwareHasKbd`	The device has a built-in keyboard.
`sysFtrNumFiveWayNavVersion`	The version of the 5-way navigation support (if any)

Hardware Features

Table 34.6 lists the feature constants that identify the hardware itself, rather than the capabilities of the hardware.

NOTE: You are strongly encouraged to check the hardware capabilities and other features before resorting to checking for specific hardware devices, manufacturers, or ROM versions. The other features, when present, provide a more reliable method of discovering the capabilities of a particular device.

Table 34.6 Hardware feature constants

Constant	Creator	Value
`sysFtrNumHwrMisc Flags`	`sysFtrCreator`	One or more flags indicating hardware capabilities. See Table 34.7 on page 723. This feature is not applicable on devices with ARM processors. On releases before Palm OS 3.5, this feature's value contains both the product ID and the hardware's capabilities.
`sysFtrNumHwrMisc FlagsExt`	`sysFtrCreator`	One or more flags indicating additional hardware capabilities. See Table 34.8 on page 724.

Table 34.6 Hardware feature constants *(continued)*

Constant	Creator	Value
		This feature is not applicable on devices with ARM processors.
`sysFtrNumOEMCompanyID`	`sysFtrCreator`	The 4-character company ID of the HAL manufacturer. The company ID is unique for each Palm OS licensee. This feature is only defined on Palm OS 3.5 and later releases. On earlier releases, the `sysFtrNumHwrMiscFlags` value contains both the product ID and the hardware's capabilities.
`sysFtrNumOEMDeviceID`	`sysFtrCreator`	The 4-character ID of the device on which Palm OS is running. There is roughly one device ID per model of device. This feature is only defined on Palm OS 3.5 and later releases. On earlier releases, the `sysFtrNumHwrMiscFlags` value contains both the product ID and the hardware's capabilities.
`sysFtrNumOEMHALID`	`sysFtrCreator`	The 4-character ID of the HAL on which Palm OS is running. Each HAL is specific to a device model and Palm OS version.

Table 34.6 Hardware feature constants *(continued)*

Constant	Creator	Value
		This feature is only defined on Palm OS 3.5 and later releases. On earlier releases, the `sysFtrNumHwrMiscFlags` value contains both the product ID and the hardware's capabilities.
`sysFtrNumOEMROM Version`	`sysFtrCreator`	A ROM system version provided by the Palm OS licensee. Used to identify patches provided by licensees.
`sysFtrNum ProcessorID`	`sysFtrCreator`	The processor type and the processor revision. See Table 34.9 on page 725.
`sysFtrNumProductID`	`sysFtrCreator`	The processor type and the processor revision. See Table 34.9 on page 725. (Note that `sysFtrNumOEMDeviceID` gives the product ID.)
`sysFtrNumROMVersion`	`sysFtrCreator`	The ROM version. You can use the `sysMakeROMVersion` macro to create a value to test against. See Appendix B and "The System Version Feature" on page 338 in the *Palm OS Programmer's Companion*, vol. I.
`sysFtrNumVendor`		Not used.

Table 34.7 defines the possible flags that could be set in the value for the `sysFtrNumHwrMiscFlags` feature constant. You can AND these flags with the value returned by `FtrGet` to see if that attribute exists. For example:

```
err = FtrGet(sysFtrCreator, sysFtrNumHwrMiscFlags, &flags);
if (!err) {
  if (flags & hwrMiscFlagHasBacklight) ...
}
```

Table 34.7 Hardware flags

Constant	Description
`hwrMiscFlagHasBacklight`	The display has a backlight.
`hwrMiscFlagHasMbdIrDA`	IrDA is present on the main board.
`hwrMiscFlagHasCardIrDA`	IrDA is present on the memory card.
`hwrMiscFlagHasBurrBrown`	Not used.
`hwrMiscFlagHasJerryHW`	The device is a Palm VII series device.
`hwrMiscFlagNoRTCBug`	Not used.
`hwrMiscFlagHas3vRef`	Not used.
`hwrMiscFlagHasAntennaSw`	Not used.
`hwrMiscFlagHasCradleDetect`	There is an A/D converter on the HotSync port used for identifying the attached device.
`hwrMiscFlagHasSWContrast`	Software contrast is supported.
`hwrMiscFlagInvertLCDForBL`	The LCD is inverted when backlight is on.
`hwrMiscFlagHasMiscFlagExt`	Further attributes are defined in the `sysFtrNumHwrMiscFlagsExt` feature value.

Table 34.8 describes the possible flags that could be set in the value for the `sysFtrNumHwrMiscFlagsExt` feature constant. You can AND these flags with the value returned by `FtrGet` to see if that attribute exists. For example:

```
err = FtrGet(sysFtrCreator, sysFtrNumHwrMiscFlagsExt,
  &flags);
if (!err) {
```

```
  if (flags & hwrMiscFlagExtHasLiIon) ...
}
```

Table 34.8 Hardware extended flags

Constant	Description
`hwrMiscFlagExtSubIDMask`	Subtype ID Mask.
`hwrMiscFlagExtHasLiIon`	The device has a Lithium Ion battery that is rechargable in the cradle.
`hwrMiscFlagExtHasRailIO`	Not used.
`hwrMiscFlagExtHasFlash`	The device has Flash ROM.
`hwrMiscFlagExtHasFParms`	The device has a Flash parameters area.
`hwrMiscFlagExt115KIrOK`	The device supports 115K IR transfers.
`hwrMiscFlagExtHasExtLCD`	Not used.
`hwrMiscFlagExtHasSWBright`	The device has software controlled brightness.
`hwrMiscFlagExtNeedsLpr`	Not used.

Table 34.9 contains the possible values returned for `sysFtrNumProcessorID` and `sysFtrNumProductID`. Use the bit mask `sysFtrNumProcessorMask` to extract the processor type from the values returned for these feature. For example:

```
UInt32 id, chip, revision;
Err err;
err = FtrGet(sysFtrCreator, sysFtrNumProcessorID, &id);
if (!err) {
  chip = id & sysFtrNumProcessorMask;
  revision = id & 0x0ffff;
  if (chip==sysFtrNumProcessor328)
    // traditional Dragonball
  else if (chip==sysFtrNumProcessorEZ)
    // Dragonball EZ
  else ...
}
```

Table 34.9 Processor values

Constant	Description
`sysFtrNumProcessor328`	Motorola 68328 (Dragonball).
`sysFtrNumProcessorARM710A`	ARM710A
`sysFtrNumProcessorARM720T`	ARM 720T
`sysFtrNumProcessorARM7TDMI`	ARM7TDMI
`sysFtrNumProcessorARM920T`	ARM920T
`sysFtrNumProcessorARM922T`	ARM922T
`sysFtrNumProcessorARM925`	ARM925
`sysFtrNumProcessorARM925T`	ARM925T
`sysFtrNumProcessorARM926EJS`	ARM926EJ-S
`sysFtrNumProcessorEZ`	Motorola 68EZ328 (Dragonball EZ).
`sysFtrNumProcessorStrongARM`	Strong ARM
`sysFtrNumProcessorSuperVZ`	Motorola 68SZ328 (Dragonball SuperVZ)
`sysFtrNumProcessorVZ`	Motorola 68VZ328 (Dragonball VZ)
`sysFtrNumProcessorx86`	The Palm OS ARM Simulator, which runs on Windows. See "Calling ARM-Native Subroutines" on page 406 in *Palm OS Programmer's Companion*, vol. I.
`sysFtrNumProcessorXscale`	X-scale

Emulator and Debugging Features

Table 34.10 lists the feature constants that provide debugging information.

Table 34.10Debugging feature constants

Constant	Creator	Value
`kPalmOSEmulator FeatureNumber`	`kPalmOSEmulator FeatureCreator`	The value is 0 if the application is running on the Palm OS Emulator rather than a real handheld.
`omFtrShowErrorsFlag`	`omFtrCreator`	A Boolean value that controls the number of error messages displayed by the Overlay Manager. Application developers may set this feature to debug an overlay database. It is automatically set in debug ROMs. See "Overlay Manager Constants" on page 870 for more information.
`sysFtrNumErrorCheck Level`	`sysFtrCreator`	The error checking level (such as `ERROR_CHECK_FULL` or `ERROR_CHECK_NONE`) with which the system was built.
`sysFtrNumGremlins SupportGlobals`	`sysFtrCreator`	A memory location that is used to intelligently support Gremlins. This feature was set only by the Progress Manager and only in Palm OS 3.2 and 3.3. It is obsolete in Palm OS 3.5 and later.

Localization Features

Table 34.11 lists the feature constants that provide information about the device's locale.

Table 34.11Localization feature constants

Constant	Creator	Value
`omFtrDefaultLocale`	`omFtrCreator`	An `LmLocaleType` constant identifying the country and language of the ROM's default locale.
`sysFtrDefaultBoldFont`	`sysFtrCreator`	The `FontID` of the default font used for bold text as specified in the ROM's locale module.
`sysFtrDefaultFont`	`sysFtrCreator`	The `FontID` of the default font for standard text, as defined in the ROM's locale module.
`sysFtrIDKeyboardActive`	`sysFtrTKeyboard`	A nonzero value indicates that the keyboard dialog is being displayed. Only used on systems with multi-byte character encodings.
`sysFtrNumCharEncodingFlags`	`sysFtrCreator`	One or more of the Character Encoding Feature Attributes, which specify the attributes of the character encoding used on the device. For example, these constants specify if the device uses only single-byte characters or has double-byte characters as well.
`sysFtrNumCountry`	`sysFtrCreator`	One of the Country Constants, identifying the default country as specified in the ROM's locale module.

Table 34.11 Localization feature constants *(continued)*

Constant	Creator	Value
`sysFtrNumEncoding`	`sysFtrCreator`	One of the Character Encoding Constants identifying the character encoding used on the device. See also "Retrieving the Character Encoding" on page 265 in the *Palm OS Programmer's Companion*, vol. I.
`sysFtrNumIntlMgr`	`sysFtrCreator`	One or more flags specifying International Manager attributes. See Table 34.12 on page 728.
`sysFtrNumLanguage`	`sysFtrCreator`	One of the Language Constants specifying the default language in the ROM's locale module.
`tsmFtrNumFlags`	`tsmFtrCreator`	One or more flags indicating the attributes of the front-end processor (FEP). The only supported attribute currently is `tsmFtrFlagsHasFep`, which is set when any FEP is present. FEPs are common on systems with multi-byte character encodings.

Table 34.12 lists the possible flags that could be set in the value for the `sysFtrNumIntlMgr` feature constant.

Table 34.12 International Manager flags

Constant	Description
`intlMgrExists`	The International Manager is installed on the device.

Table 34.12International Manager flags *(continued)*

Constant	Description
`intlMgrStrict`	The International Manager does strict checking and produces more error messages. This flag is set for debugging purposes.
`intlMgrBestFit`	The `TxtConvertEncoding()` function can use the `charEncodingDestBestFitFlag`. See the description of that function for more information.

Miscellaneous Features

Table 34.13 lists the feature constants that provide information about the other features that may be part of Palm OS.

Table 34.13Miscellaneous feature constants

Constant	Creator	Value
`sndFtrIDVersion`	`sysFileCSoundMgr`	The version number of the Sound Streaming API. Used to determine if the Sound Stream Feature Set is present.
`sysFtrIDOEMSysHideBatteryGauge`	`sysFtrTOEMSys`	If defined, the Launcher does not display the battery gauge in its title bar. This feature is only defined on devices such as phones that have some other place to display the information.
`sysFtrNumAccessorTrapPresent`	`sysFtrCreator`	A nonzero value indicates that the Palm OS Glue Library running on a Palm OS Garnet release should call certain functions through PACE rather than access structures directly.

Table 34.13 Miscellaneous feature constants *(continued)*

Constant	Creator	Value
`sysFtrNumNotifyMgrVersion`	`sysFtrCreator`	The version number of the Notification Manager API. Used to determine if the Notification Feature Set is present.
`sysFtrNumWinVersion`	`sysFtrCreator`	The version of the Window Manager. If 4 or greater, the High-Density Display Feature Set is present. Version 5 adds QVGA support.

Feature Manager Functions and Macros

FtrGet Function

Purpose Get a feature.

Declared In `FeatureMgr.h`

Prototype

```
Err FtrGet (UInt32 creator, UInt16 featureNum,
    UInt32 *valueP)
```

Parameters

→ *creator*
Creator ID, which must be registered with PalmSource, Inc. This is usually the same as the creator ID for the application that owns this feature.

→ *featureNum*
Feature number of the feature.

← *valueP*
Value of the feature is returned here.

Returns Returns 0 if no error, or `ftrErrNoSuchFeature` if the specified feature number doesn't exist for the specified creator.

Comments The value of the feature is application-dependent.

See Also `FtrSet()`

FtrGetByIndex Function

Purpose Get a feature by index.

Declared In `FeatureMgr.h`

Prototype

```
Err FtrGetByIndex (UInt16 index,
   Boolean romTable,
   UInt32 *creatorP, UInt16 *numP,
   UInt32 *valueP)
```

Parameters

→ *index*
Index of feature.

→ *romTable*
If `true`, index into ROM table; otherwise, index into RAM table.

← *creatorP*
Feature creator is returned here.

← *numP*
Feature number is returned here.

← *valueP*
Feature value is returned here.

Returns Returns 0 if no error, or `ftrErrNoSuchFeature` if the index is out of range.

Comments This function is intended for system use only. It is used by shell commands. Most applications don't need it.

Until the caller gets back `ftrErrNoSuchFeature`, it should pass indices for each table (ROM, RAM) starting at 0 and incrementing. Note that in Palm OS 3.1 and higher, the RAM feature table serves the entire system. At system startup, the values in the ROM feature table are copied into the RAM feature table.

FtrPtrFree Function

Purpose Release memory previous allocated with `FtrPtrNew()`.

Declared In `FeatureMgr.h`

Prototype

```
Err FtrPtrFree (UInt32 creator,
   UInt16 featureNum)
```

Parameters

→ *creator*
The creator ID for the feature.

→ *featureNum*
Feature number of the feature.

Returns Returns 0 if no error, or `ftrErrNoSuchFeature` if an error occurs.

Comments This function unregisters the feature before freeing the memory associated with it.

Compatibility Implemented only if 3.1 New Feature Set is present.

FtrPtrNew Function

Purpose Allocate feature memory.

Declared In `FeatureMgr.h`

Prototype

```
Err FtrPtrNew (UInt32 creator, UInt16 featureNum,
   UInt32 size, void **newPtrP)
```

Parameters

→ *creator*
Creator ID, which must be registered with PalmSource, Inc. This is usually the same as the creator ID for the application that owns this feature.

→ *featureNum*
Feature number of the feature.

→ *size*
Size in bytes of the temporary memory to allocate. The maximum chunk size is 64K.

← *newPtrP*
Pointer to the memory chunk is returned here.

Returns Returns 0 if no error, `memErrInvalidParam` if the value of `size` is 0, or `memErrNotEnoughSpace` if there is not enough space to allocate a chunk of the specified size.

Comments This function allocates a chunk of memory and stores a pointer to that chunk in the feature table. The same pointer is returned in `newPtrP`. The memory chunk remains allocated and locked until the next system reset or until you free the chunk with `FtrPtrFree()`.

`FtrPtrNew` is useful if you want quick, efficient access to data that persists from one invocation of the application to the next. `FtrPtrNew` stores values on the storage heap rather than the dynamic heap, where free space is often extremely limited. The disadvantage to using feature memory is that writing to storage memory is slower than writing to dynamic memory.

NOTE: Starting with Palm OS 3.5, `FtrPtrNew` allows allocating chunks larger than 64k. Do keep in mind standard issues with allocating large chunks of memory: there might not be enough contiguous space, and it can impact system performance.

You can obtain the pointer to the chunk using `FtrGet()`. To write to the chunk, you *must* use `DmWrite()` because the chunk is in the storage heap, not the dynamic heap.

For example, if you allocate a memory chunk in this way:

```
FtrPtrNew(appCreator,
    myFtrMemFtr, 32, &ftrMem);
```

You can later access that memory and write to it using the following:

```
void *data;
if (!FtrGet(appCreator,
    myFtrMemFtr, (UInt32*)&data))
  DmWrite(data, 0, &someVal, sizeof(someVal));
```

Compatibility Implemented only if 3.1 New Feature Set is present.

See Also `FtrPtrResize()`

FtrPtrResize Function

Purpose Resize feature memory.

Declared In `FeatureMgr.h`

Prototype

```
Err FtrPtrResize (UInt32 creator,
   UInt16 featureNum, UInt32 newSize,
   void **newPtrP)
```

Parameters

→ *creator*
The creator ID for the feature.

→ *featureNum*
Feature number of the feature.

→ *newSize*
New size in bytes for the chunk.

← *newPtrP*
Pointer to the memory chunk is returned here.

Returns Returns 0 if no error, or `ftrErrNoSuchFeature` if the specified feature number doesn't exist for the specified creator, `memErrInvalidParam` if `newSize` is 0, or `memErrNotEnoughSpace` if there's not enough free space available to allocate a chunk of that size.

Comments Use this function to resize a chunk of memory previously allocated by `FtrPtrNew()`.

This function may move the chunk to a new location in order to resize it, so it is important to use the pointer returned by this function when accessing the memory chunk. The pointer in the feature table is automatically updated to be the same as the pointer returned by this function.

If this function fails, the old memory pointer still exists and its data is unchanged.

Compatibility Implemented only if 3.1 New Feature Set is present.

See Also `MemHandleResize()`

FtrSet Function

Purpose Set a feature.

Declared In `FeatureMgr.h`

Prototype

```
Err FtrSet (UInt32 creator, UInt16 featureNum,
   UInt32 newValue)
```

Parameters

→ *creator*
Creator ID, which must be registered with PalmSource, Inc. This is usually the same as the creator ID for the application that owns this feature.

→ *featureNum*
Feature number for this feature.

→ *newValue*
New value.

Returns Returns 0 if no error, or `memErrNotEnoughSpace` if the feature table must be resized to add a new feature and no space is available.

Comments The value of the feature is application-dependent.

A feature that you define in this manner remains defined until the next system reset or until you explicitly undefine the feature with `FtrUnregister()`.

See Also `FtrGet()`, `FtrPtrNew()`

FtrUnregister Function

Purpose Unregister a feature.

Declared In `FeatureMgr.h`

Prototype

```
Err FtrUnregister (UInt32 creator,
   UInt16 featureNum)
```

Parameters

→ *creator*
Creator ID for the feature.

→ *featureNum*
Feature number of the feature.

Returns Returns 0 if no error, or `ftrErrNoSuchFeature` if the specified feature number doesn't exist for the specified creator.

35

File Streaming

This chapter provides reference material for the file streaming API.

- File Streaming Constants
- File Streaming Functions
- File Streaming Error Codes

The header file `FileStream.h` declares the API that this chapter describes. For more information on file streaming, see the chapter "Files and Databases" in the *Palm OS Programmer's Companion*, vol. I.

File Streaming Constants

Primary Open Mode Constants

This section lists constants passed in the `openMode` parameter to the `FileOpen()` function. These constants specify the mode in which a file stream is opened.

For each file stream, you must pass to the `FileOpen()` function only one of the primary mode selectors listed.

`fileModeReadOnly`
: Open for read-only access

`fileModeReadWrite`
: Open/create for read/write access, discarding any previous version of stream

`fileModeUpdate`
: Open/create for read/write, preserving previous version of stream if it exists

`fileModeAppend`
: Open/create for read/write, always writing to the end of the stream

Secondary Open Mode Constants

You can use the | operator (bitwise inclusive OR) to append to a primary mode selector one or more of the secondary mode selectors listed below.

`fileModeDontOverwrite`
: Prevents `fileModeReadWrite` from discarding an existing stream having the same name; may only be specified together with `fileModeReadWrite`

`fileModeLeaveOpen`
: Leave stream open when application quits. Most applications should not use this option.

`fileModeExclusive`
: No other application can open the stream until the application that opened it in this mode closes it.

`fileModeAnyTypeCreator`
: Accept any type/creator when opening or replacing an existing stream. Normally, the `FileOpen()` function opens only streams having the specified creator and type. Setting this option enables the `FileOpen` function to open streams having a type or creator other than those specified.

`fileModeTemporary`
: Delete the stream automatically when it is closed. For more information, see Comment section of `FileOpen()` function description.

File Streaming Functions

FileClearerr Function

Purpose Clear I/O error status, end of file error status, and last error.

Declared In `FileStream.h`

Prototype
```
Err FileClearerr (FileHand stream)
```

Parameters → *stream*
: Handle to open stream.

Returns 0 if no error, or a `fileErr` code if an error occurs. See the section "File Streaming Error Codes" for more information.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileGetLastError()`, `FileRewind()`

FileClose Function

Purpose Close the file stream and destroy its handle. If the stream was opened with `fileModeTemporary`, it is deleted upon closing.

Declared In `FileStream.h`

Prototype

```
Err FileClose (FileHand stream)
```

Parameters

→ *`stream`*
Handle to open stream.

Returns 0 if no error, or a `fileErr` code if an error occurs. See the section "File Streaming Error Codes" for more information.

Compatibility Implemented only if 3.0 New Feature Set is present.

FileControl Function

Purpose Perform the operation specified by the op parameter on the `stream` file stream.

Declared In `FileStream.h`

Prototype

```
Err FileControl (FileOpEnum op, FileHand stream,
    void *valueP, Int32 *valueLenP)
```

Parameters

op
The operation to perform, and its associated formal parameters. See the Comments section for a list of possible values.

→ *`stream`*
Open stream handle if required for file stream operation.

↔ *`valueP`*
Pointer to value or buffer, as required. This parameter is defined by the selector passed as the value of the op parameter. For details, see the Comments section.

↔ *valueLenP*
: Pointer to value or buffer, as required. This parameter is defined by the selector passed as the value of the op parameter. For details, see the Comments section.

Returns Returns either a value defined by the selector passed as the argument to the op parameter, or an error code resulting from the requested operation. For details, see the Comments section.

Comments Normally, you do not call the `FileControl()` function yourself; it is called for you by most of the other file streaming functions and macros to perform common file streaming operations. You can call `FileControl()` yourself to enable specialized read modes.

`fileOpNone`
: No-op.

`fileOpDestructiveReadMode`
: Enter destructive read mode, and rewind stream to its beginning. Once in this mode, there is no turning back: stream's contents after closing (or crash) are undefined.

  Destructive read mode deletes blocks as data are read, thus freeing storage automatically. Once in destructive read mode, you cannot re-use the file stream—the contents of the stream are undefined after it is closed or after a crash.

  Writing to files opened without write access or those that are in destructive read state is not allowed; thus, you cannot call the `FileWrite()`, `FileSeek()`, or `FileTruncate()` functions on a stream that is in destructive read mode. One exception to this rule applies to streams that were opened in "write + append" mode and then switched into destructive read state. In this case, the `FileWrite()` function can append data to the stream, but it also preserves the current

stream position so that subsequent reads pick up where they left off (you can think of this as a pseudo-pipe).

ARGUMENTS:

`stream` = open stream handle

`valueP` = `NULL`

`valueLenP` = `NULL`

RETURNS:

zero on success;

`fileErr`... on error

`fileOpGetEOFStatus`

Get end-of-file status (like C runtime's `feof`) (err = `fileErrEOF`). Indicates end of file condition. Use `FileClearerr()` to clear this error status.

ARGUMENTS:

`stream` = open stream handle

`valueP` = `NULL`

`valueLenP` = `NULL`

RETURNS:

zero if not end of file;

non-zero if end of file

`fileOpGetLastError`

Get error code from last operation on stream, and clear the last error code value. Doesn't change status of EOF or I/O errors —use `FileClearerr()` to reset all error codes.

ARGUMENTS:

`stream` = open stream handle

`valueP` = `NULL`

`valueLenP` = `NULL`

RETURNS:

Error code from last file stream operation

`fileOpClearError`

Clear I/O and EOF error status and last error.

ARGUMENTS:

`stream` = open stream handle

`valueP` = `NULL`

`valueLenP` = `NULL`

RETURNS:

zero on success; `fileErr`... on error

`fileOpGetIOErrorStatus`

Get I/O error status (like C runtime's `ferror`). Use `FileClearerr()` to clear this error status.

ARGUMENTS:

`stream` = open stream handle

`valueP` = `NULL`

`valueLenP` = `NULL`

RETURNS:

zero if not I/O error;

non-zero if I/O error is pending.

`fileOpGetCreatedStatus`

Find out whether file was created by `FileOpen()` function

ARGUMENTS:

`stream` = open stream handle

`valueP` = Pointer to Boolean

`valueLenP` = Pointer to `Int32` variable set to sizeof(Boolean)

RETURNS:

zero on success; `fileErr`... on error. The Boolean variable will be set to non-zero if the file was created.

`fileOpGetOpenDbRef`

Get the open database reference (handle) of the underlying database that implements the stream (`NULL` if none); this is needed for performing Palm OS-specific operations on the

underlying database, such as changing or getting creator and type, version, backup/reset bits, and so on.

ARGUMENTS:

`stream` = open stream handle

`valueP` = Pointer to `DmOpenRef` variable

`valueLenP` = Pointer to Int32 variable set to sizeof(DmOpenRef)

RETURNS:

zero on success; `fileErr`... on error. The `DmOpenRef` variable will be set to the file's open db reference that may be passed to Data Manager calls;

WARNING: Do not make any changes to the data of the underlying database -- doing so will corrupt the file stream.

`fileOpFlush`
Flush any cached data to storage.

ARGUMENTS:

`stream` = open stream handle

`valueP` = `NULL`

`valueLenP` = `NULL`

RETURNS:

zero on success; `fileErr`... on error;

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileClearerr()`, `FileEOF()`, `FileError()`, `FileFlush()`, `FileGetLastError()`, `FileRewind()`

FileDelete Function

Purpose Deletes the specified file stream from the specified card. Only a closed stream may be passed to this function.

Declared In `FileStream.h`

Prototype

```
Err FileDelete (UInt16 cardNo, const Char *nameP)
```

Parameters

`cardNo`
: Card on which the file stream to delete resides. Currently, no Palm OS® devices support multiple cards, so this value must be 0.

`nameP`
: String that is the name of the stream to delete.

Returns 0 if no error, or a `fileErr` code if an error occurs. See the section "File Streaming Error Codes" for more information.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileOpen()`

FileDmRead Macro

Purpose Reads data from a file stream into a chunk, record, or resource residing in a database.

Declared In `FileStream.h`

Prototype

```
#define FileDmRead(stream, startOfDmChunkP,
   destOffset, objSize, numObj, errP)
```

Parameters

→ *`stream`*
: Handle to open stream.

→ *`startOfDmChunkP`*
: Pointer to beginning of chunk, record or resource residing in a database.

`destOffset`
: Offset from `startOfDmChunkP` (base pointer) to the destination area (must be >= 0).

`objSize`
: Size of each stream object to read.

`numObj`
Number of stream objects to read.

↔ *`errP`*
Pointer to variable that is to hold the error code returned by this function. Pass `NULL` to ignore. See the section "File Streaming Error Codes" for more information.

Returns The number of whole objects that were read—note that the number of objects actually read may be less than the number requested.

Comments When the number of objects actually read is less than the number requested, you may be able to determine the cause of this result by examining the return value of the `errP` parameter or by calling the `FileGetLastError()` function. If the cause is insufficient data in the stream to satisfy the full request, the current stream position is at end-of-file and the "end of file" indicator is set. If a non-`NULL` pointer was passed as the value of the `errP` parameter when the `FileDmRead` function was called and an error was encountered, `*errP` holds a non-zero error code when the function returns. In addition, the `FileError()` and `FileEOF()` functions may be used to check for I/O errors.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileRead()`, `FileError()`, `FileEOF()`

FileEOF Function

Purpose Get end-of-file status (err = `fileErrEOF` indicates end of file condition).

Declared In `FileStream.h`

Prototype
```
Err FileEOF (FileHand stream)
```

Parameters → *`stream`*
Handle to open stream.

Returns 0 if *not* end of file; non-zero if end of file. See the section "File Streaming Error Codes" for more information.

Comments This function's behavior is similar to that of the `feof` function provided by the C programming language runtime library.

Use `FileClearerr()` to clear the I/O error status.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileClearerr()`, `FileGetLastError()`, `FileRewind()`

FileError Function

Purpose Get I/O error status.

Declared In `FileStream.h`

Prototype

```
Err FileError (FileHand stream)
```

Parameters → *stream*
Handle to open stream.

Returns 0 if no error, and non-zero if an I/O error indicator has been set for this stream. See the section "File Streaming Error Codes" for more information.

Comments This function's behavior is similar to that of the C programming language's `ferror` runtime function.

Use `FileClearerr()` to clear the I/O error status.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileClearerr()`, `FileGetLastError()`, `FileRewind()`

FileFlush Function

Purpose Flush cached data to storage.

Declared In `FileStream.h`

Prototype

```
Err FileFlush (FileHand stream)
```

Parameters → *stream*
Handle to open stream.

Returns 0 if no error, or a `fileErr` code if an error occurs. See the section "File Streaming Error Codes" for more information.

Comments It is not always necessary to call this function explicitly—certain operations flush the contents of a stream automatically; for example, streams are flushed when they are closed. Because this function's behavior is similar to that of the `fflush` function provided by the C programming language runtime library, you only need to call it

explicitly under circumstances similar to those in which you would call `fflush` explicitly.

Compatibility Implemented only if 3.0 New Feature Set is present.

FileGetLastError Function

Purpose Get error code from last operation on file stream, and clear the last error code value (will not change end of file or I/O error status -- use `FileClearerr()` to reset all error codes)

Declared In `FileStream.h`

Prototype

```
Err FileGetLastError (FileHand stream)
```

Parameters

→ *stream*
Handle to open stream.

Returns Error code returned by the last file stream operation. See the section "File Streaming Error Codes" for more information.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileClearerr()`, `FileEOF()`, `FileError()`

FileOpen Function

Purpose Open existing file stream or create an open file stream for I/O in the mode specified by the `openMode` parameter.

Declared In `FileStream.h`

Prototype

```
FileHand FileOpen (UInt16 cardNo,
   const Char *nameP, UInt32 type,
   UInt32 creator,
   UInt32 openMode, Err *errP)
```

Parameters

`cardNo`
Card on which the file stream to open resides. Currently, no Palm Powered™ devices support multiple cards, so this value must be 0.

→ *nameP*
Pointer to text string that is the name of the file stream to open or create. This value must be a valid name—no wildcards allowed, must not be `NULL`.

`type`
: File type of stream to open or create. Pass `0` for wildcard, in which case `sysFileTFileStream` is used if the stream needs to be created and `fileModeTemporary` is not specified. If type is `0` and `fileModeTemporary` is specified, then `sysFileTTemp` is used for the file type of the stream this function creates.

`creator`
: Creator of stream to open or create. Pass `0` for wildcard, in which case the current application's creator ID is used for the creator of the stream this function creates.

`openMode`
: Mode in which to open the file stream. You must specify only one primary mode selector. Additionally, you can use the `|` operator (bitwise inclusive OR) to append one or more secondary mode selectors to the primary mode selector. See "Primary Open Mode Constants" and "Secondary Open Mode Constants" for the list of possible values.

↔ *`errP`*
: Pointer to variable that is to hold the error code returned by this function. Pass `NULL` to ignore. See the section "File Streaming Error Codes" for a list of error codes.

Returns

If successful, returns a handle to an open file stream; otherwise, returns `0`.

In some cases, on some platforms, `FileOpen()` returns a non-zero value when it has failed to open a file; thus, it is always a good idea to check the `errP` parameter value to determine if an error has occurred.

Comments

The `fileModeReadOnly`, `fileModeReadWrite`, `fileModeUpdate`, and `fileModeAppend` modes are mutually exclusive—pass only one of them to the `FileOpen` function!

When the `fileModeTemporary` open mode is used and the file type passed to `FileOpen` is `0`, the `FileOpen` function uses `sysFileTTemp` (defined in `SystemMgr.rh`) for the file type, as recommended. In future versions of Palm OS, this configuration will enable the automatic cleanup of undeleted temporary files after a system crash. Automatic post-crash cleanup is not implemented in current versions of Palm OS.

To open a file stream even if it has a different type and creator than specified, pass the `fileModeAnyTypeCreator` selector as a flag in the `openMode` parameter to the `FileOpen()` function.

The `fileModeLeaveOpen` mode is an esoteric option that most applications should not use. It may be useful for a library that needs to open a stream from the current application's context and keep it open even after the current application quits. By default, Palm OS automatically closes all databases that were opened in a particular application's context when that application quits. The `fileModeLeaveOpen` option overrides this default behavior.

Compatibility Implemented only if 3.0 New Feature Set is present.

FileRead Macro

Purpose Reads data from a stream into a buffer. Do not use this macro to read data into a chunk, record or resource residing in a database—you must use the `FileDmRead()` macro for such operations.

Declared In `FileStream.h`

Prototype

```
#define FileRead(stream, bufP, objSize, numObj,
    errP)
```

Parameters

→ *stream*
: Handle to open stream.

→ *bufP*
: Pointer to beginning of buffer into which data is read

`objSize`
: Size of each stream object to read.

`numObj`
: Number of stream objects to read.

↔ *errP*
: Pointer to variable that is to hold the error code returned by this function. Pass `NULL` to ignore. See the section "File Streaming Error Codes" for a list of error codes.

Returns The number of whole objects that were read—note that the number of objects actually read may be less than the number requested.

Comments Do not use this macro to read data into a chunk, record or resource residing in a database—you must use the `FileDmRead()` macro for such operations.

When the number of objects actually read is fewer than the number requested, you may be able to determine the cause of this result by examining the return value of the `errP` parameter or by calling the `FileGetLastError()` function. If the cause is insufficient data in the stream to satisfy the full request, the current stream position is at end-of-file and the "end of file" indicator is set. If a non-`NULL` pointer was passed as the value of the `errP` parameter when the `FileRead` function was called and an error was encountered, `*errP` holds a non-zero error code when the function returns. In addition, the `FileError()` and `FileEOF()` functions may be used to check for I/O errors.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileDmRead()`

FileRewind Function

Purpose Reset position marker to beginning of stream and clear all error codes.

Declared In `FileStream.h`

Prototype

```
Err FileRewind (FileHand stream)
```

Parameters

→ *stream*
Handle to open stream.

Returns 0 if no error, or a `fileErr` code if an error occurs. See the section "File Streaming Error Codes" for more information.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileSeek()`, `FileTell()`, `FileClearerr()`, `FileEOF()`, `FileError()`, `FileGetLastError()`

FileSeek Function

Purpose Set current position within a file stream, extending the stream as necessary if it was opened with write access.

Declared In `FileStream.h`

Prototype

```
Err FileSeek (FileHand stream, Int32 offset,
    FileOriginEnum origin)
```

Parameters

→ *`stream`*
: Handle to open stream.

`offset`
: Position to set, expressed as the number of bytes from origin. This value may be positive, negative, or `0`.

`origin`
: Describes the origin of the position change. Possible values are:

  `fileOriginBeginning`
  : From the beginning (first data byte of file).

  `fileOriginCurrent`
  : From the current position.

  `fileOriginEnd`
  : From the end of file (one position beyond last data byte).

Returns 0 if no error, or a `fileErr` code if an error occurs. See the section "File Streaming Error Codes" for more information.

Comments Attempting to seek beyond end-of-file in a read-only stream results in an I/O error.

This function's behavior is similar to that of the `fseek` function provided by the C programming language runtime library.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileRewind()`, `FileTell()`

FileTell Function

Purpose Retrieves the current position and, optionally, file size, of a stream.

Declared In `FileStream.h`

Prototype

```
Int32 FileTell (FileHand stream, Int32 *fileSizeP,
    Err *errP)
```

Parameters

→ *stream*
Handle to open stream.

↔ *fileSizeP*
Pointer to variable that holds value describing size of stream in bytes when this function returns. Pass `NULL` to ignore.

↔ *errP*
Pointer to variable that is to hold the error code returned by this function. Pass `NULL` to ignore. See the section "File Streaming Error Codes" for a list of possible error codes.

Returns If successful, returns current position, expressed as an offset in bytes from the beginning of the stream. If an error was encountered, returns `-1` as a signed long integer.

Comments The `FileTell` function can return the size of the input stream; as such, it provides some of the functionality of the standard C library `stat` function. Note, however, that unlike the `stat` function, `FileTell` requires that the file be open.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileRewind()`, `FileSeek()`

FileTruncate Function

Purpose Truncate the file stream to a specified size; not allowed on streams open in destructive read mode or read-only mode.

Declared In `FileStream.h`

Prototype

```
Err FileTruncate (FileHand stream, Int32 newSize)
```

Parameters

→ *stream*
Handle of open stream.

`newSize`
New size; must not exceed current stream size.

Returns 0 if no error, or a `fileErr` code if an error occurs. See the section "File Streaming Error Codes" for a list of possible error codes.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FileTell()`

FileWrite Function

Purpose Write data to a stream.

Declared In `FileStream.h`

Prototype

```
Int32 FileWrite (FileHand stream,
   const void *dataP, Int32 objSize,
   Int32 numObj,
   Err *errP)
```

Parameters

→ *stream*
Handle to open stream.

→ *dataP*
Pointer to buffer holding data to write.

`objSize`
Size of each stream object to write; must be ≥ `0`.

`numObj`
Number of stream objects to write.

↔ *errP*
Optional pointer to variable that holds the error code returned by this function. Pass `NULL` to ignore. See the section "File Streaming Error Codes" for a list of possible error codes.

Returns The number of whole objects that were written—note that the number of objects actually written may be less than the number requested. Should available storage be insufficient to satisfy the entire request, as much of the requested data as possible is written to the stream, which may result in the last object in the stream being incomplete.

Comments Writing to files opened without write access or those that are in destructive read state is not allowed; thus, you cannot call the `FileWrite()`, `FileSeek()`, or `FileTruncate()` functions on a stream that is in destructive read mode. One exception to this rule

applies to streams that were opened in "write + append" mode and then switched into destructive read state. In this case, the `FileWrite` function can append data to the stream, but it also preserves the current stream position so that subsequent reads pick up where they left off (you can think of this as a pseudo-pipe).

Compatibility Implemented only if 3.0 New Feature Set is present.

File Streaming Error Codes

This section lists all error codes returned by the file streaming functions.

Error Code	Value	Meaning
`fileErrMemErr`	(`fileErrorClass`|1)	Out of memory error
`fileErrInvalidParam`	(`fileErrorClass`|2)	Invalid parameter value passed
`fileErrCorruptFile`	(`fileErrorClass`|3)	Alleged stream is corrupted, invalid, or not a stream
`fileErrNotFound`	(`fileErrorClass`|4)	Couldn't find the stream
`fileErrTypeCreatorMismatch`	(`fileErrorClass`|5)	Type and/or creator not what was specified
`fileErrReplaceError`	(`fileErrorClass`|6)	Couldn't replace existing stream
`fileErrCreateError`	(`fileErrorClass`|7)	Couldn't create new stream
`fileErrOpenError`	(`fileErrorClass`|8)	Generic open error
`fileErrInUse`	(`fileErrorClass`|9)	Stream couldn't be opened or deleted because it is in use

Error Code	Value	Meaning
`fileErrReadOnly`	(`fileErrorClass`|10)	Couldn't open in write mode because existing stream is read-only
`fileErrInvalidDescriptor`	(`fileErrorClass`|11)	Invalid file descriptor (`FileHandle`)
`fileErrCloseError`	(`fileErrorClass`|12)	Error closing the stream
`fileErrOutOfBounds`	(`fileErrorClass`|13)	Attempted operation went out of bounds of the stream
`fileErrPermissionDenied`	(`fileErrorClass`|14)	Couldn't write to a stream open for read-only access
`fileErrIOError`	(`fileErrorClass`|15)	Generic I/O error
`fileErrEOF`	(`fileErrorClass`|16)	End-of-File error
`fileErrNotStream`	(`fileErrorClass`|17)	Attempted to open an entity that is not a stream

36

Float Manager

This chapter provides reference material for the Float Manager API as follows:

- Float Manager Data Structures
- Float Manager Functions

The Float Manager API is declared in the header file `FloatMgr.h`. For more information on the Float Manager, see the section "Floating-Point" in the *Palm OS Programmer's Companion*, vol. I.

Float Manager Data Structures

FlpCompDouble Struct

Purpose

Float Manager functions accept and require values of type `FlpDouble`. The `FlpCompDouble` union allows you to declare values that can be interpreted either as a double or as an `FlpDouble`. As well, this union contains fields that provide easy access to the component parts of the double-precision floating-point number.

Prototype

```
typedef union {
  double d;
  FlpDouble fd;
  UInt32 ul[2];
  FlpDoubleBits fdb;
} FlpCompDouble
```

Fields

`d`

Provides access to the value as a `double`.

`fd`

Provides access to the value as a `FlpDouble`, which can be passed to or received from many Float Manager functions.

`ul`
: Provides access to the value as two long integers.

`fdb`
: Provides access to specific fields.

FlpDoubleBits Struct

Purpose

This structure provides direct access to the component parts of an IEEE-754 double-precision floating-point number. Use the `FlpCompDouble` union to convert numbers of type `double` to and from `FlpDoubleBits`.

Prototype

```
typedef struct {
  UInt32 sign : 1;
  Int32 exp : 11;
  UInt32 manH : 20;
  UInt32 manL;
} FlpDoubleBits
```

Fields

`sign`
: The sign bit. You can also use the `FlpGetSign()` macro to obtain the sign bit, and the `FlpNegate()`, `FlpSetNegative()`, and `FlpSetPositive()` macros to set the sign bit.

`exp`
: The bits that make up the exponent. You can also use the `FlpGetExponent()` macro to obtain the exponent value.

`manH`
: The most-significant 20 bits of the mantissa.

`manL`
: The least-significant 32 bits of the mantissa.

Float Manager Functions

FlpAToF Function

Purpose Convert a null-terminated ASCII string to a 64-bit floating-point number. The string must have the format:

```
[+|-][digits][.][digits][e|E[+|-][digits]]
```

Declared In `FloatMgr.h`

Prototype `FlpDouble FlpAToF (const Char *s)`

Parameters → *s*

Pointer to the string to be converted.

Returns Returns the value of the string as a floating-point number.

Comments The mantissa of the number is limited to 32 bits.

This function is close to being compatible with the ISO C library function `atof`. `atof` requires the form:

```
[+|-]digits[.][digits][(e|E)[+|-]digits]
```

In order to maintain backward compatibility with the Float Manager in Palm OS 1.0 (which could be used up to, but not including, Palm OS 4.0), this function considers *all* of the "digits" sections to be optional. Here's a table showing the ISO and Palm OS behavior with some sample strings:

String	ISO	>= Palm OS 4.0	< Palm OS 4.0[1]	Notes
"+"	+0	+0	+0	
".3"	0.3	0.3	0.3	
"0.3e123"	0.3e123	0.3e123	0.3e12	The old Float Manager only allowed a 1 or 2 digit exponent.
"+1"	1	1	+0	The old Float Manager doesn't allow a leading '+' sign.

String	ISO	>= Palm OS 4.0	< Palm OS 4.0[1]	Notes
"1e+2"	1e2	1e2	1	The old Float Manager doesn't allow a '+' sign in the exponent.
"0.3E3"	0.3e3	0.3e3	0.3	The old Float Manager doesn't allow a capital 'E' to mark the exponent.
"4294967297"	4294967297	4294967297	1	The old Float Manager uses an unsigned long and wraps around.

1. Using the old Float Manager documented in Appendix C, "1.0 Float Manager," on page 2461.

Unlike `atof`, `FlpAToF` doesn't accept leading white-space characters and it doesn't accept decimal point characters other than '`.`'.

Compatibility Implemented only if 2.0 New Feature Set is present. GCC users must use `FlpBufferAToF()` instead of this function.

See Also `FlpFToA()`

FlpBase10Info Function

Purpose Extract detailed information on the base 10 form of a floating-point number: the base 10 mantissa, exponent, and sign.

Declared In `FloatMgr.h`

Prototype

```
Err FlpBase10Info (FlpDouble a, UInt32 *mantissaP,
   Int16 *exponentP, Int16 *signP)
```

Parameters

→ *a*
The floating-point number.

← *mantissaP*
The base 10 mantissa.

← *exponentP*
The base 10 exponent.

← *signP*
The sign: 1 if the number is negative, 0 otherwise.

Returns Returns 0 if no error, or `flpErrOutOfRange` if the supplied floating-point number is either not a number (NaN) or is infinite.

Comments The mantissa is normalized so it contains at least 8 significant digits when printed as an integer value.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `FlpGetExponent()`, `FlpGetSign()`

FlpBufferAToF Function

Purpose Convert a null-terminated ASCII string to a floating-point number. The string must be in the format: `[-]x[.]yyyyyyyy[e[-]zz]`

Declared In `FloatMgr.h`

Prototype

```
void FlpBufferAToF (FlpDouble *result,
   const Char *s)
```

Parameters

← *result*
Pointer to the structure into which the return value is placed.

→ *s*
Pointer to the null-terminated ASCII string to be converted.

Returns Returns the value of the string as a floating-point number.

Comments See `FlpAToF()` for a complete description of this function.

Compatibility Implemented only if 2.0 New Feature Set is present. Because the Palm OS ABI was not well-specified in this area, GCC by default implemented structure return differently from the compiler used to build the ROM. As a result, GCC users must use this function instead of `FlpAToF()`. CodeWarrior users can use either function; they are binary compatible.

FlpBufferCorrectedAdd Function

Purpose Adds two floating-point numbers and corrects for least-significant-bit errors when the result should be zero but is instead very close to zero.

Declared In `FloatMgr.h`

Prototype

```
void FlpBufferCorrectedAdd (FlpDouble *result,
    FlpDouble firstOperand,
    FlpDouble secondOperand,
    Int16 howAccurate)
```

Parameters

← *result*
: Pointer to the structure into which the return value is placed.

→ *firstOperand*
: The first of the two numbers to be added.

→ *secondOperand*
: The second of the two numbers to be added.

→ *howAccurate*
: The smallest difference in exponents that won't force the result to zero. The value returned from this function is forced to zero if the difference between exponents in the smaller of the two operands and the result exceeds this value. Supply a value of zero for this parameter to obtain the default level of accuracy (which is equivalent to a `howAccurate` value of 48).

Returns Returns the calculated result.

Comments See `FlpCorrectedAdd()` for a complete description of this function.

Compatibility Implemented only if 2.0 New Feature Set is present. Because the Palm OS ABI was not well-specified in this area, GCC by default implemented structure return differently from the compiler used to build the ROM. As a result, GCC users must use this function instead of `FlpCorrectedAdd()`. CodeWarrior users can use either function; they are binary compatible.

FlpBufferCorrectedSub Function

Purpose Subtracts two floating-point numbers and corrects for least-significant-bit errors when the result should be zero but is instead very close to zero.

Declared In `FloatMgr.h`

Prototype

```
void FlpBufferCorrectedSub (FlpDouble *result,
    FlpDouble firstOperand,
    FlpDouble secondOperand,
    Int16 howAccurate)
```

Parameters

← *result*
Pointer to the structure into which the return value is placed.

→ *firstOperand*
The value from which `secondOperand` is to be subtracted.

→ *secondOperand*
The value to subtract from `firstOperand`.

→ *howAccurate*
The smallest difference in exponents that won't force the result to zero. The value returned from this function is forced to zero if the difference between exponents in the smaller of the two operands and the result exceeds this value. Supply a value of zero for this parameter to obtain the default level of accuracy (which is equivalent to a `howAccurate` value of 48).

Returns Returns the calculated result.

Comments See `FlpCorrectedSub()` for a complete description of this function.

Compatibility Implemented only if 2.0 New Feature Set is present. Because the Palm OS ABI was not well-specified in this area, GCC by default implemented structure return differently from the compiler used to build the ROM. As a result, GCC users must use this function instead of `FlpCorrectedSub()`. CodeWarrior users can use either function; they are binary compatible.

FlpCorrectedAdd Function

Purpose Adds two floating-point numbers and corrects for least-significant-bit errors when the result should be zero but is instead very close to zero.

Declared In `FloatMgr.h`

Prototype

```
FlpDouble FlpCorrectedAdd (FlpDouble firstOperand,
    FlpDouble secondOperand, Int16 howAccurate)
```

Parameters

→ *`firstOperand`*
The first of the two numbers to be added.

→ *`secondOperand`*
The second of the two numbers to be added.

→ *`howAccurate`*
The smallest difference in exponents that won't force the result to zero. The value returned from `FlpCorrectedAdd` is forced to zero if, when the exponent of the result of the addition is subtracted from the exponent of the smaller of the two operands, the difference exceeds the value specified for `howAccurate`. Supply a value of zero for this parameter to obtain the default level of accuracy (which is equivalent to a `howAccurate` value of 48).

Returns Returns the calculated result.

Comments Adding or subtracting a large number and a small number produces a result similar in magnitude to the larger number. Adding or subtracting two numbers that are similar in magnitude can, depending on their signs, produce a result with a very small exponent (that is, a negative exponent that is large in magnitude). If the difference between the result's exponent and that of the operands is close to the number of significant bits expressible by the mantissa, it is quite possible that the result should in fact be zero.

There also exist cases where it may be useful to retain accuracy in the low-order bits of the mantissa. For instance: 99999999 + 0.00000001 - 99999999. However, unless the fractional part is an exact (negative) power of two, it is doubtful that what few bits of mantissa that are available will be enough to properly represent the fractional value. In this example, the 99999999 requires 26 bits, leaving 26 bits for the .00000001; this guarantees inaccuracy after the subtraction.

The problem arises from the difficulty in representing decimal fractions such as 0.1 in binary. After about three successive additions or subtractions, errors begin to appear in the least significant bits of the mantissa. If the value represented by the most significant bits of the mantissa is then subtracted away, the least significant bit error is normalized and becomes the actual result—when in fact the result should be zero.

This problem is only an issue for addition and subtraction.

Compatibility Implemented only if 2.0 New Feature Set is present. GCC users must use `FlpBufferCorrectedAdd()` instead of this function.

See Also `FlpCorrectedSub()`

FlpCorrectedSub Function

Purpose Subtracts two floating-point numbers and corrects for least-significant-bit errors when the result should be zero but is instead very close to zero.

Declared In `FloatMgr.h`

Prototype

```
FlpDouble FlpCorrectedSub (FlpDouble firstOperand,
    FlpDouble secondOperand, Int16 howAccurate)
```

Parameters

→ *firstOperand*
The value from which `secondOperand` is to be subtracted.

→ *secondOperand*
The value to subtract from `firstOperand`.

→ *howAccurate*
The smallest difference in exponents that won't force the result to zero.The value returned from `FlpCorrectedSub` is forced to zero if, when the exponent of the result of the subtraction is subtracted from the exponent of the smaller of the two operands, the difference exceeds the value specified for `howAccurate`. Supply a value of zero for this parameter to obtain the default level of accuracy (which is equivalent to a `howAccurate` value of 48).

Returns Returns the calculated result.

Comments See the comments for `FlpCorrectedAdd()`.

Compatibility Implemented only if 2.0 New Feature Set is present. GCC users must use `FlpBufferCorrectedSub()` instead of this function.

FlpFToA Function

Purpose Convert a floating-point number to a null-terminated ASCII string in exponential format: `[-]x.yyyyyyye[-]zz`

Declared In `FloatMgr.h`

Prototype

```
Err FlpFToA (FlpDouble a, Char *s)
```

Parameters

→ *a*
Floating-point number.

← *s*
Pointer to buffer to contain the ASCII string.

Returns Returns 0 if no error, or `flpErrOutOfRange` if the supplied value is infinite or is not a number. In this case, the buffer is set to the string "INF", "-INF", or "NaN" as appropriate.

Comments If the supplied floating-point number is zero, the returned string doesn't use the exponential format but is simply "0".

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `FlpAToF()`

FlpGetExponent Macro

Purpose Returns the exponent of a 64-bit floating-point value. The returned value has the bias applied, so it ranges from -1023 to +1024.

Declared In `FloatMgr.h`

Prototype

```
#define FlpGetExponent (x)
```

Parameters

→ *x*
The value from which the exponent is to be extracted.

Returns Returns a `UInt32` containing the exponent of the specified value.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `FlpBase10Info()`, `FlpGetSign()`

FlpGetSign Macro

Purpose Returns the sign of a 64-bit floating-point value.

Declared In `FloatMgr.h`

Prototype
```
#define FlpGetSign (x)
```

Parameters → *x*
The value from which the sign bit is to be extracted.

Returns Returns a `UInt32` with a nonzero value if the specified value is negative, and with a zero value if it is positive.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `FlpBase10Info()`, `FlpGetExponent()`, `FlpNegate()`, `FlpSetNegative()`, `FlpSetPositive()`

FlpIsZero Macro

Purpose Returns whether the specified 64-bit floating-point value is zero.

Declared In `FloatMgr.h`

Prototype
```
#define FlpIsZero (x)
```

Parameters → *x*
The value for which the sign bit is desired.

Returns Returns a `UInt32` with a nonzero value if the specified value is zero, and with a zero value if the specified value is other than zero.

Compatibility Implemented only if 2.0 New Feature Set is present.

FlpNegate Macro

Purpose Changes the sign bit of a 64-bit floating-point number.

Declared In `FloatMgr.h`

Prototype
```
#define FlpNegate (x)
```

Parameters → *x*
The value in which the sign bit is to be changed.

Returns Returns a 64-bit floating-point value which is the negative of the value specified by `x`.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `FlpGetSign()`, `FlpSetNegative()`, `FlpSetPositive()`

FlpSetNegative Macro

Purpose Ensures that a 64-bit floating-point number is negative.

Declared In `FloatMgr.h`

Prototype

```
#define FlpSetNegative (x)
```

Parameters → *x*

The value that is to be forced negative.

Returns If the supplied 64-bit floating-point value is negative, that value is returned unchanged. If the supplied value is positive, the negative of that value is returned.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `FlpGetSign()`, `FlpNegate()`, `FlpSetPositive()`

FlpSetPositive Macro

Purpose Ensures that a 64-bit floating-point number is positive.

Declared In `FloatMgr.h`

Prototype

```
#define FlpSetPositive (x)
```

Parameters → *x*

The value that is to be forced positive.

Returns If the supplied 64-bit floating-point value is positive, that value is returned unchanged. If the supplied value is negative, its absolute value is returned.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `FlpGetSign()`, `FlpNegate()`, `FlpSetNegative()`

FlpVersion Function

Purpose Returns the version number of the Float Manager.

Declared In `FloatMgr.h`

Prototype

```
UInt32 FlpVersion (void)
```

Parameters `None.`

Returns Returns the version number of the Float Manager. The current version is represented by the constant `flpVersion`, which is defined in `FloatMgr.h`.

Compatibility Implemented only if 2.0 New Feature Set is present.

37

Fonts

This chapter provides the following information regarding font support:

- Font Data Structures
- Font Constants
- Font Resources
- Font Functions

The header files `Font.h` and `FontSelect.h` declare the API that this chapter describes. For more information on fonts, see Chapter 8, "Text," on page 259 of the *Palm OS Programmer's Companion*, vol. I.

Font Data Structures

FontCharInfoPtr Typedef

Purpose The `FontCharInfoPtr` type points to a `FontCharInfoType` structure.

Prototype

```
typedef FontCharInfoType *FontCharInfoPtr;
```

FontCharInfoType Struct

Purpose The `FontCharInfoType` structure defines an entry in the offset/width table for a font.

WARNING! Palm, Inc. does not support or provide backward compatibility for the `FontCharInfoType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct FontCharInfoTag {
  Int8 offset;
  Int8 width;
} FontCharInfoType;
```

Fields

`offset`
: This value is not currently used and must be set to 0.

`width`
: The exact width in pixels of the glyph. You can retrieve this information using the function `FntWCharWidth()` or `FntCharWidth()`.

FontDensityType Struct

Purpose

The `FontDensityType` structure defines an entry in the `densities` array in the `FontTypeV2` structure. The `densities` array specifies the location of each set of glyphs within an extended font resource.

WARNING! Palm, Inc. does not support or provide backward compatibility for the `FontDensityType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct FontDensityTag {
  Int16 density;
  UInt32 glyphBitsOffset;
} FontDensityType;
```

Fields

`density`
: Either `kDensityLow`, `kDensityOneAndAHalf`, or `kDensityDouble`.

`glyphBitsOffset`
: Offset in bytes from the beginning of the font data to the start of the font image for this density.

Compatibility

This structure is only defined if the High-Density Display Feature Set is present.

FontID Enum

Purpose

The `FontID` enum specifies the IDs of available fonts. A font can either be a system-defined font or an application-defined font. You can obtain the ID of the current font using `FntGetFont()` and change the font using `FntSetFont()`.

Prototype

```
enum fontID {
  stdFont = 0x00,
  boldFont,
  largeFont,
  symbolFont,
  symbol11Font,
  symbol7Font,
  ledFont,
  largeBoldFont,
  fntAppCustomBase = 0x80
};
typdef enum fontID FontID;
```

Constants

`stdFont`
: A small standard font used to display user input. This font is small to display as much text as possible.

`boldFont`
: Same size as `stdFont` but bold for easier reading. Used for text labels in the user interface.

`largeFont`
: A larger font provided as an alternative for users who find the standard font too small to read.

`symbolFont`
: Contains many special characters such as arrows, shift indicators, and so on.

`symbol11Font`
: Contains the check boxes, the large left arrow, and the large right arrow.

`symbol7Font`
: Contains the up and down arrows used for the repeating button scroll arrows and the dimmed version of the same arrows.

`ledFont`
: Contains the numbers 0 through 9, –, ., and the comma (,). Used by the Calculator application for its numeric display.

`largeBoldFont`
: In Palm OS® 3.0 and later only. Same size as `largeFont` but bold.

`fntAppCustomBase`
: The first available ID for application-defined fonts.

FontPtr Typedef

Purpose The `FontPtr` type defines a pointer to a `FontType` structure.

Prototype

```
typedef FontType *FontPtr;
```

FontType Struct

Purpose The `FontType` structure defines a font resource's header. The fields in this structure give general information about the font. Following the structure are several tables that Palm OS uses to draw the font on the screen. See "Font Resource" on page 779 for more information about the font resource.

WARNING! Palm, Inc. does not support or provide backward compatibility for the `FontType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct FontTag {
  Int16 fontType;
  Int16 firstChar;
  Int16 lastChar;
  Int16 maxWidth;
  Int16 kernMax;
  Int16 nDescent;
  Int16 fRectWidth;
  Int16 fRectHeight;
  Int16 owTLoc;
```

```
  Int16 ascent;
  Int16 descent;
  Int16 leading;
  Int16 rowWords;
} FontType;
```

Fields

`fontType`
: A mask providing the general characteristics of the font. When creating an application-defined font resource, use 0x9000.

`firstChar`
: Character code of first glyph in the font.

`lastChar`
: Character code of last glyph in the font.

`maxWidth`
: The maximum width in pixels of any glyph. In Palm OS, there is currently no difference between this field and `fRectWidth`.

`kernMax`
: This value is not currently used and must be set to 0.

`nDescent`
: This value is not currently used and must be set to 0.

`fRectWidth`
: A metric of the font image. In Palm OS, this metric is equivalent to the maximum width in pixels of any glyph in the font. Use `FntAverageCharWidth()` to obtain this value.

`fRectHeight`
: The height, including ascenders and descenders, of the glyphs in this font. Use `FntCharHeight()` to obtain this value.

`owTLoc`
: The offset in 16-bit words from this field to the first byte of the offset/width table. The offset/width table is a table of `FontCharInfoType` structures giving the width of each character in the font. Do not access the offset/width table directly. Use `FntWCharWidth()` or `FntCharWidth()` instead.

`ascent`
: The distance in pixels from the top of the font rectangle to its baseline. Use `FntBaseLine()` to obtain this value.

`descent`
: The distance in pixels from the baseline to the bottom of the font rectangle. Use `FntDescenderHeight()` to obtain this value.

`leading`
: The font's leading, which is the vertical space between lines of text, in pixels. This field is unused in Palm OS and must be set to 0. If your font requires a leading value, add blank space to the bottom of each of your glyphs. The `FntLineHeight()` function returns the size of the font's character cell plus the leading.

`rowWords`
: The number of 16-bit words stored for each row of a glyph's bitmap where `fRectHeight` is the number of rows.

FontTypeV2 Struct

Purpose

The `FontTypeV2` structure defines the header for an extended font resource, which contains a separate set of glyphs for each screen density. Currently the only supported densities are `kDensityLow` and `kDensityDouble`. See "Extended Font Resource" on page 782 for more information.

WARNING! Palm, Inc. does not support or provide backward compatibility for the `FontCharTypeV2` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct FontTagV2 {
  Int16 fontType;
  Int16 firstChar;
  Int16 lastChar;
  Int16 maxWidth;
  Int16 kernMax;
  Int16 nDescent;
  Int16 fRectWidth;
```

```
  Int16 fRectHeight;
  Int16 owTLoc;
  Int16 ascent;
  Int16 descent;
  Int16 leading;
  Int16 rowWords;
  Int16 version;
  Int16 densityCount;
  FontDensityType densities[0];
} FontTypeV2;
```

NOTE: All pixel values given in the fields below are in terms of the single-density font data.

Fields

`fontType`
: A mask providing the general characteristics of the font. When creating an application-defined extended font resource, use the value `fntExtendedFormatMask | 0x9000`.

`firstChar`
: Character code of first glyph in the font.

`lastChar`
: Character code of last glyph in the font.

`maxWidth`
: The maximum width in pixels of any glyph. In Palm OS, there is currently no difference between this field and `fRectWidth`.

`kernMax`
: This value is not currently used and must be set to 0.

`nDescent`
: This value is not currently used and must be set to 0.

`fRectWidth`
: A metric of the font image. In Palm OS, this metric is equivalent to the maximum width in pixels of any glyph in the font. Use `FntAverageCharWidth()` to obtain this value.

`fRectHeight`
: The height, including ascenders and descenders, of the glyphs in this font. Use `FntCharHeight()` to obtain this value.

`owTLoc`
: The offset in 16-bit words from this field to the first byte of the offset/width table. The offset/width table is a table of `FontCharInfoType` structures giving the width of each character in the font. Do not access the offset/width table directly. Use `FntWCharWidth()` or `FntCharWidth()` instead.

`ascent`
: The distance in pixels from the top of the font rectangle to its baseline. Use `FntBaseLine()` to obtain this value.

`descent`
: The distance in pixels from the baseline to the bottom of the font rectangle. Use `FntDescenderHeight()` to obtain this value.

`leading`
: The font's leading, which is the vertical space between lines of text, in pixels. This field is unused in Palm OS and must be set to 0. If your font requires a leading value, add blank space to the bottom of each of your glyphs. The `FntLineHeight()` function returns the size of the font's character cell plus the leading.

`rowWords`
: The number of 16-bit words stored for each row of a glyph's bitmap where `fRectHeight` is the number of rows.

`version`
: The version of the extended font resource. This value should be set to 1.

`densityCount`
: The number of entries in the `densities` array.

`densities`
: An array of one or more `FontDensityType` structures identifying the glyphs for each supported density.

Compatibility

This structure is only defined if the High-Density Display Feature Set is present.

Font Constants

Constant	Value	Description
`checkboxFont`	`symbol11Font`	A convenience constant that points to the font containing the checkbox bitmap.
`fntMissingChar`	–1	The value used for a character that does not have a definition in the current font. The missing character symbol is usually an open rectangle.
`fntExtendedFormatMask`	0x0200	A constant used for the `fontType` field of a font to indicate that it is an extended font resource.
`fntTabChrWidth`	20	The width of the tab character in pixels.

Font Resources

Font Resource

The font resource (`'NFNT'`) represents a version 1 single-density font. This resource is the same as the Macintosh `'NFNT'` resource with some restrictions. It contains a header followed by several tables that provide information about each glyph in the font.

Figure 37.1 shows how the font resource is laid out in memory. Table 37.1 describes each table within the font resource.

Figure 37.1 Font resource ('NFNT')

Table 37.1 Font resource description

Field	Description
FontType header	Contains general information about the glyphs in the font. See `FontType`.
Font image	A raw bitmap image containing the packed character glyphs from left to right (see Figure 37.2 on page 782). This part of the resource tells Palm OS how to draw each character in the font. The height of the image is `fRectHeight` and the size is `rowWords * 2 * fRectHeight`.

Table 37.1 Font resource description *(continued)*

Field	Description
	Place glyphs sequentially in order of increasing character code. Leave at least a one-pixel wide vertical column of space to the right of each image so that there is space between characters when Palm OS draws text on the screen. If your font requires leading, leave horizontal space at the bottom of the characters as well. The font image must end with the glyph for the missing character symbol.
Bitmap location table	A table of 16-bit words that specify the location of each glyph's entry in the font image. The location is specified as the bit offset from the start of the image to the glyph in the first row of the font image. The last entry in the table contains the offset of the column after the last bitmap. (See Figure 37.2 on page 782.) If you have skipped characters within an encoding, for each glyph that is missing, specify the same value for its location as the entry for the next glyph in the table.
Offset/width table	A table that specifies how wide each glyph in the font is. On Macintosh systems, this table also specifies how each glyph kerns. Palm OS does not support kerning, as the offset value is ignored. Each entry in the offset/width table is two bytes long. The first byte should be 0, and the second byte should contain the glyph width, which must be greater than or equal to 0. If the glyph at this index does not have a bitmap in the font image, the values should be –1 and –1.

Figure 37.2 shows an example of the font image for a font that defines glyphs for four characters (A, B, C, and the missing

character symbol) and the portion of the bitmap location table that provides the offsets for these characters. The last entry in the bitmap location table is the offset to the column after the last bitmap, or 0x0014.

Figure 37.2 Font image and bitmap location table

```
Font image:  |-----|-----|-----|-----|
             |-----|-----|-----|-----|
             |-----|-----|-----|-----|
             |-##--|###--|-##--|#####|
             |#--#-|#--#-|#--#-|#---#|
             |#--#-|###--|#----|#---#|
             |####-|#--#-|#----|#---#|
             |#--#-|#--#-|#--#-|#---#|
             |#--#-|###--|-##--|#####|
             |-----|-----|-----|-----|
             |-----|-----|-----|-----|
Bitmap location table values: 0x0000  0x0005  0x000A  0x000F  0x0014
```

Extended Font Resource

The extended font resource (`'nfnt'`) defines a font that supports multiple screen densities. The High-Density Display Feature Set supports two or more screen densities. As shown in Figure 37.3, the extended font resource is essentially:

- A `FontTypeV2` header giving all general information about the glyphs in the font. All metrics are in terms of the low-density version of the font.
- Tables for the low-density font. See "Font Resource" on page 779 for a description of these tables.
- The font image (set of glyphs) for each density specified by the font.

NOTE: It is *not* required that an extended font resource contain a font image for the low-density font. It may contain a font image only for a double-density font, for example. All of the metrics and the font tables must be given in standard coordinates, as if there were a low-density version of the font present in the resource.

Compatibility

This resource is only defined if the High-Density Display Feature Set is present. 1.5x density fonts are only supported if the 1.5x Display Feature Set is present.

Figure 37.3 Extended font resource

Font Functions

FntAverageCharWidth Function

Purpose Gets the *maximum* character width in pixels of the current font.

Declared In `Font.h`

Prototype
```
Int16 FntAverageCharWidth (void)
```

Parameters None.

Returns Returns the *maximum* character width (in pixels).

Comments This function returns the value of the `fRectHeight` field in the `FontType` structure for the current font. Because Palm OS does not support kerning, this value is the maximum width in pixels rather than the average width.

FntBaseLine Function

Purpose Gets the distance from the top of the character cell to the baseline for the current font.

Declared In `Font.h`

Prototype
```
Int16 FntBaseLine (void)
```

Parameters None.

Returns Returns the ascent of the font (in pixels).

FntCharHeight Function

Purpose Gets the character height of the current font including accents and descenders.

Declared In `Font.h`

Prototype
```
Int16 FntCharHeight (void)
```

Parameters None.

Returns Returns the height of the characters in the current font, expressed in pixels.

FntCharsInWidth Function

Purpose Finds the length in bytes of the characters from a specified string that fit within a passed width.

Declared In `Font.h`

Prototype

```
void FntCharsInWidth (Char const *string,
    Int16 *stringWidthP, Int16 *stringLengthP,
    Boolean *fitWithinWidth)
```

Parameters

→ *`string`*
: A pointer to the character string.

↔ *`stringWidthP`*
: The maximum width to allow (in pixels). Upon return, contains the actual width allowed. Note that this value does not include any trailing spaces or tabs, which are stripped by this function.

↔ *`stringLengthP`*
: The maximum length of text to allow, in bytes (assumes current font). Upon return, contains the number of bytes of text that can appear within the width. Note that this value does not include any trailing space or tabs, which are stripped by this function.

← *`fitWithinWidth`*
: Upon return, `false` if the string is considered truncated, `true` if it isn't.

Returns `Returns nothing.`

Comments Spaces and tabs at the end of a string are ignored and removed. If the string fits within the specified width after spaces and tabs are removed, the `fitWithinWidth` value contains `true`. Characters after a carriage return are ignored, and the string is considered truncated.

This function is specifically designed for the code used to draw text fields. Consider using `FntWidthToOffset()` in your application code instead, particularly if you do not want the special processing of trailing spaces, tabs, and carriage returns.

FntCharsWidth Function

Purpose Gets the width of the specified character string. The **missing character symbol** (an open rectangle) is substituted for any character that does not exist in the current font.

Declared In `Font.h`

Prototype

```
Int16 FntCharsWidth (Char const *chars,
    Int16 len)
```

Parameters

→ *chars*
Pointer to a string of characters.

→ *len*
Length in bytes of the string.

Returns Returns the width of the string, in pixels.

Comments Like all functions that work with strings, this function returns correct results for strings with multi-byte characters as well as strings with only single-byte characters.

See Also `FntCharWidth()`

FntCharWidth Function

Purpose Gets the width of the specified character. If the specified character does not exist within the current font, the missing character symbol is substituted.

Declared In `Font.h`

Prototype

```
Int16 FntCharWidth (Char ch)
```

Parameters

→ *ch*
Character whose width is needed.

Returns Returns the width of the specified character (in pixels).

Comments `FntCharWidth` works with single-byte characters only. To determine the pixel width of a single-byte character or a multi-byte character, use `FntWCharWidth()` instead of this function in Palm OS 4.0 and higher. Alternatively, for compatibility with earlier versions of Palm OS, link with the PalmOSGlue library and call

`FntGlueWCharWidth`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `FntCharsWidth()`

FntDefineFont Function

Purpose Makes a custom font available to your application.

Declared In `Font.h`

Prototype

```
Err FntDefineFont (FontID font, FontPtr fontP)
```

Parameters

→ *font*
: A value greater than or equal to `fntAppFontCustomBase` that identifies the custom font to the system. Values less than that are reserved for system use. Note that font IDs are 8-bit unsigned values and so must be less than 256. See `FontID`.

→ *fontP*
: Pointer to the custom font resource to be used by this function. This resource must remain locked until the calling application undefines the custom font or quits.

Returns

`errNone`
: No error

`memErrNotEnoughSpace`
: Insufficient dynamic heap space

Comments The custom font is available only when the application that called this function is running; when the application quits, the custom font is uninstalled automatically.

The font this function specifies is not available at build time; as a result, some UI elements—labels, for example—cannot determine their bounds automatically as they do when using the built-in fonts.

Before you use this function, you must load the font resource from the database and obtain a pointer to it. See "Creating Custom Fonts" on page 284 of the *Palm OS Programmer's Companion*, vol. I for more information.

Compatibility Implemented only if 3.0 New Feature Set is present.

Palm OS 4.0 and later do not allow a `NULL` value for the `fontP` parameter; therefore, you cannot undefine a font using

`FntDefineFont(myFontID, NULL)`. Earlier versions of Palm OS allowed the `NULL` value. Note that there is usually no need to undefine a font.

Palm OS 3.0 and 3.1 have problems with font ID values greater than 131. These problems are fixed in later releases of Palm OS.

See Also FontSelect(), FntSetFont()

FntDescenderHeight Function

Purpose Gets the height of a character's descender in the current font. The height of a descender is the distance between the baseline and the bottom of the character cell.

Declared In `Font.h`

Prototype

```
Int16 FntDescenderHeight (void)
```

Parameters None.

Returns Returns the height of a descender, expressed in pixels.

FntGetFont Function

Purpose Gets the font ID of the current font.

Declared In `Font.h`

Prototype

```
FontID FntGetFont (void)
```

Parameters None.

Returns Returns the ID of the current font.

Comments The current font is the font stored in the draw state. It is used when drawing characters directly onto the screen. Most user interface elements, such as fields, tables, labels, and buttons, do not use the current font.

See Also `FntSetFont()`, `FntGetFontPtr()`, `FontID`

FntGetFontPtr Function

Purpose Gets a pointer to the current font.

Declared In `Font.h`

Prototype

```
FontPtr FntGetFontPtr (void)
```

Parameters None.

Returns Returns a pointer to the current font.

Comments The current font is the font stored in the draw state. It is used when drawing characters directly onto the screen. Most user interface elements, such as fields, tables, labels, and buttons, do not use the current font.

See Also `FntGetFont()`

FntGetScrollValues Function

Purpose Gets the values needed to update a scroll bar based on a specified string and the position within the string.

Declared In `Font.h`

Prototype

```
void FntGetScrollValues (Char const *chars,
   UInt16 width, UInt16 scrollPos, UInt16 *linesP,
   UInt16 *topLine)
```

Parameters

→ *chars*
: A null-terminated string.

→ *width*
: The width of a line of text in the display, given in pixels.

→ *scrollPos*
: The byte offset of the first character displayed on the topmost line.

← *linesP*
: Number of lines required to display the string.

← *topLine*
: The line of text that is the topmost visible line. Line numbering starts with 0.

Returns Returns nothing. Stores the number of lines of text in `linesP` and the top visible line in `topLine`.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `FldGetScrollValues()`

FntIsAppDefined Macro

Purpose Returns `true` if the font is defined by the application or `false` if it is defined by the system.

Declared In `Font.h`

Prototype

```
#define FntIsAppDefined (fnt)
```

Parameters → *fnt*

The `FontID` of a font.

Returns Boolean that indicates if the font is an application-defined font. Returns `true` if application-defined, `false` if system-defined.

FntLineHeight Function

Purpose Gets the height of a line in the current font. The height of a line is the height of the character cell plus the space between lines (the external leading).

Declared In `Font.h`

Prototype

```
Int16 FntLineHeight (void)
```

Parameters None.

Returns Returns the height in pixels of a line in the current font.

FntLineWidth Function

Purpose Gets the width of the specified line of text, taking tab characters into account. The function assumes that the characters passed are left-aligned and that the first character in the string is the first character

drawn on a line. In other words, this routine doesn't work for characters that don't start at the beginning of a line.

Declared In `Font.h`

Prototype

```
Int16 FntLineWidth (Char const *pChars,
   UInt16 length)
```

Parameters

→ *pChars*
Pointer to a string of characters.

→ *length*
Length in bytes of the string.

Returns Returns the line width (in pixels).

FntSetFont Function

Purpose Sets the current font.

Declared In `Font.h`

Prototype

```
FontID FntSetFont (FontID font)
```

Parameters

→ *font*
ID of the font to make the current font.

Returns Returns the ID of the previous font.

Comments If the specified font ID is invalid, this function sets the current font to `stdFont`.

The current font is the font stored in the draw state. It is used when drawing characters directly onto the screen. Most user interface elements, such as fields, tables, labels, and buttons, do not use the current font. To set the font for one of these elements, check the API for that element. If the element's API doesn't have a function to set the font programmatically, check the PalmOSGlue library.

See Also `FntGetFont()`

FntWCharWidth Function

Purpose Gets the width of the specified character. If the specified character does not exist within the current font, the missing character symbol is substituted.

Declared In `Font.h`

Prototype

```
Int16 FntWCharWidth (WChar iChar)
```

Parameters

→ *`iChar`*
Character whose width is needed.

Returns Returns the width of the specified character (in pixels).

Comments `FntWCharWidth` works with both single-byte characters and multi-byte characters. However, you should always pass a `WChar` variable to this function rather than a `Char` to avoid sign extension problems on values 0x80 and higher.

Compatibility Implemented only if 4.0 New Feature Set is present. If you want to use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `FntGlueWCharWidth`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `FntCharWidth()`

FntWidthToOffset Function

Purpose Given a pixel position, gets the offset of the character displayed at that location.

Declared In `Font.h`

Prototype

```
Int16 FntWidthToOffset (Char const *pChars,
   UInt16 length, Int16 pixelWidth,
   Boolean *leadingEdge, Int16 *truncWidth)
```

Parameters

→ *`pChars`*
Pointer to the character string.

→ *`length`*
The length in bytes of `pChars`.

→ *`pixelWidth`*
A horizontal pixel offset from the beginning of the string.

← *leadingEdge*
Set to `true` if the pixel position `pixelWidth` falls on the left side of the character. Pass `NULL` for this parameter if you don't need this information.

← *truncWidth*
The width of the text (in pixels) up to but not including the returned offset. Pass `NULL` for this parameter if you don't need this information.

Returns Returns the byte offset into `pChars` of the character that contains the pixel offset `pixelWidth`. If `pixelWidth` is past the right edge of the string, the function returns the byte offset past the last character in `pChars`, and `truncWidth` contains the width required to display the entire string.

Compatibility Implemented only if 3.1 New Feature Set is present. If you want to use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `FntGlueWidthToOffset`. For more information, see Chapter 80, "PalmOSGlue Library."

FntWordWrap Function

Purpose Given a string, determines how many bytes of text can be displayed within the specified width with a line break at a tab or space character.

Declared In `Font.h`

Prototype

```
UInt16 FntWordWrap (Char const *chars,
    UInt16 maxWidth)
```

Parameters

→ *chars*
A pointer to a null-terminated string.

→ *maxWidth*
The maximum line width in pixels.

Returns Returns the length of the line, in bytes. If the entire string cannot be displayed within `maxWidth`, the value that this function returns specifies the offset where the line should be broken, which is typically following a space, tab, or line-feed character.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `FldWordWrap()`

FntWordWrapReverseNLines Function

Purpose Word wraps a text string backwards by the number of lines specified. The character position of the start of the first line and the number of lines that are actually word wrapped are returned.

Declared In `Font.h`

Prototype

```
void FntWordWrapReverseNLines
    (Char const *const chars, UInt16 maxWidth,
    UInt16 *linesToScrollP, UInt16 *scrollPosP)
```

Parameters

→ *chars*
A pointer to a null-terminated string.

→ *maxWidth*
The maximum line width in pixels.

↔ *linesToScrollP*
The number of lines to scroll. Upon return, contains the number of lines that were scrolled.

↔ *scrollPosP*
The byte offset of the first character displayed on the topmost line. Upon return, contains the first character after wrapping.

Returns Returns nothing. Stores the first character after wrapping and the number of lines scrolled in `scrollPosP` and `linesToScrollP`.

Compatibility Implemented only if 2.0 New Feature Set is present.

FontSelect Function

Purpose Displays a dialog from which the user can choose one of the system-supplied fonts, and returns a `FontID` value representing the user's choice.

Declared In `FontSelect.h`

Prototype

```
FontID FontSelect (FontID fontID)
```

Parameters → *`fontID`*

A `FontID` value specifying the font to be highlighted as the default choice in the dialog box that this function displays. This value must be one of the following system-supplied constants:

`stdFont`
: Standard plain text font.

`boldFont`
: Bold version of `stdFont`.

`largeFont`
: A large plain text font.

`largeBoldFont`
: Larger version of `boldFont`.

Returns Returns a `FontID` value representing the font that the user chose.

Comments When your application launches for the first time, it should determine the system's default font. The default font varies based on locale. You can use `FntGlueGetDefaultFontID()` from the PalmOSGlue library to determine the default font as follows:

```
fntID =
FntGlueGetDefaultFontID(defaultSystemFont);
```

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also FntGetFont(), FntSetFont()

38

Graffiti Manager

This chapter provides reference material for the Graffiti® Manager. The Graffiti Manager API is declared in the header file `Graffiti.h`.

For more information on the Graffiti Manager, see "The Graffiti Manager" on page 60 of the *Palm OS Programmer's Companion*, vol. I.

Graffiti Manager Functions

GrfAddMacro Function

Purpose Add a macro to the macro list.

Declared In `Graffiti.h`

Prototype

```
Err GrfAddMacro (const Char *nameP,
   UInt8 *macroDataP, UInt16 dataLen)
```

Parameters

`nameP`
Name of macro.

`macroDataP`
Data of macro.

`dataLen`
Size of macro data in bytes.

Returns Returns 0 if no error; returns `grfErrNoMacros`, `grfErrMacroPtrTooSmall`, `dmErrNotValidRecord`, `dmErrWriteOutOfBounds` if an error occurs.

See Also `GrfGetMacro()`, `GrfGetMacroName()`, `GrfDeleteMacro()`

GrfAddPoint Function

Purpose Add a point to the Graffiti point buffer.

Declared In `Graffiti.h`

Prototype
```
Err GrfAddPoint (PointType *pt)
```

Parameters `pt`
Pointer to point buffer.

Returns Returns 0 if no error; returns `grfErrPointBufferFull` if an error occurs.

Compatibility If the Graffiti 2 Feature Set is present, this function has no effect. It simply returns 0.

See Also `GrfFlushPoints()`

GrfCleanState Function

Purpose Remove any temporary shifts from the dictionary state.

Declared In `Graffiti.h`

Prototype
```
Err GrfCleanState (void)
```

Parameters None

Returns Returns 0 if no error, or `grfErrNoDictionary` if an error occurs.

Compatibility If the Graffiti 2 Feature Set is present, this function has no effect. It simply returns 0.

See Also `GrfInitState()`

GrfDeleteMacro Function

Purpose Delete a macro from the macro list.

Declared In `Graffiti.h`

Prototype
```
Err GrfDeleteMacro (UInt16 index)
```

Parameters `index`
Index of the macro to delete.

Returns Returns 0 if no error, or `grfErrNoMacros`, `grfErrMacroNotFound` if an error occurs.

See Also `GrfAddMacro()`

GrfFilterPoints Function

Purpose Filter the points in the Graffiti point buffer.

Declared In `Graffiti.h`

Prototype

```
Err GrfFilterPoints (void)
```

Parameters None.

Returns Always returns 0.

Compatibility If the Graffiti 2 Feature Set is present, this function has no effect. It simply returns 0.

See Also `GrfMatch()`

GrfFindBranch Function

Purpose Locate a branch in the Graffiti dictionary by flags.

Declared In `Graffiti.h`

Prototype

```
Err GrfFindBranch (UInt16 flags)
```

Parameters

`flags`
: Flags of the branch you're searching for.

Returns Returns 0 if no error, or `grfErrNoDictionary` or `grfErrBranchNotFound` if an error occurs.

Compatibility If the Graffiti 2 Feature Set is present, this function has no effect. It simply returns 0.

See Also `GrfCleanState()`, `GrfInitState()`

GrfFlushPoints Function

Purpose Dispose of all points in the Graffiti point buffer.

Declared In `Graffiti.h`

Prototype

```
Err GrfFlushPoints (void)
```

Parameters None.

Returns Always returns 0.

Compatibility If the Graffiti 2 Feature Set is present, this function has no effect. It simply returns 0.

See Also `GrfAddPoint()`

GrfGetAndExpandMacro Function

Purpose Look up and expand a macro in the current macros.

Declared In `Graffiti.h`

Prototype

```
Err GrfGetAndExpandMacro (Char *nameP,
   UInt8 *macroDataP, UInt16 *dataLenP)
```

Parameters

`nameP`
: Name of macro to look up.

`macroDataP`
: Macro contents returned here.

`dataLenP`
: On entry, size of `macroDataP` buffer; on exit, number of bytes in macro data.

Returns Returns 0 if no error, or `grfErrNoMacros` or `grfErrMacroNotFound` if an error occurs.

See Also `GrfAddMacro()`, `GrfGetMacro()`

GrfGetGlyphMapping Function

Purpose Look up a glyph in the dictionary and return the text.

Declared In `Graffiti.h`

Prototype

```
Err GrfGetGlyphMapping (UInt16 glyphID,
   UInt16 *flagsP, void *dataPtrP,
   UInt16 *dataLenP,
   UInt16 *uncertainLenP)
```

Parameters

`glyphID`
Glyph ID to look up.

`flagsP`
Returned dictionary flags.

`dataPtrP`
Where returned text goes.

`dataLenP`
On entry, size of `dataPtrP`; on exit, number of bytes returned.

`uncertainLenP`
Return number of uncertain characters in text.

Returns Returns 0 if no error, or `grfErrNoDictionary` or `grfErrNoMapping` if an error occurs.

Compatibility If the Graffiti 2 Feature Set is present, this function is deprecated and always returns `grfErrBadParam`.

See Also `GrfMatch()`

GrfGetMacro Function

Purpose Look up a macro in the current macros.

Declared In `Graffiti.h`

Prototype

```
Err GrfGetMacro (Char *nameP, UInt8 *macroDataP,
   UInt16 *dataLenP)
```

Parameters

`nameP`
Name of macro to lookup.

`macroDataP`
Macro contents returned here.

`dataLenP`
On entry: size of `macroDataP` buffer. On exit: number of bytes in macro data.

Returns Returns 0 if no error or `grfErrNoMacros`, `grfErrMacroNotFound`.

See Also `GrfAddMacro()`

GrfGetMacroName Function

Look up a macro name by index.

Declared In `Graffiti.h`

Prototype
```
Err GrfGetMacroName (UInt16 index, Char *nameP)
```

Parameters

`index`
Index of macro.

`nameP`
Name returned here.

Returns Returns 0 if no error, or `grfErrNoMacros` or `grfErrMacroNotFound` if an error occurs.

See Also `GrfAddMacro()`, `GrfGetMacro()`

GrfGetNumPoints Function

Purpose Return the number of points in the point buffer.

Declared In `Graffiti.h`

Prototype
```
Err GrfGetNumPoints (UInt16 *numPtsP)
```

Parameters

`numPtsP`
Returned number of points.

Returns Always returns 0.

Compatibility If the Graffiti 2 Feature Set is present, this function has no effect. It simply returns 0.

See Also `GrfAddPoint()`

GrfGetPoint Function

Purpose Return a point out of the Graffiti point buffer.

Declared In `Graffiti.h`

Prototype

```
Err GrfGetPoint (UInt16 index, PointType *pointP)
```

Parameters

`index`
Index of the point to get.

`pointP`
Returned point.

Returns Returns 0 if no error, or `grfErrBadParam` if an error occurs.

Compatibility If the Graffiti 2 Feature Set is present, this function is deprecated and always returns `grfErrBadParam`.

See Also GrfAddPoint(), GrfGetNumPoints()

GrfGetState Function

Purpose Return the current shift state.

Declared In `Graffiti.h`

Prototype

```
Err GrfGetState (Boolean *capsLockP,
   Boolean *numLockP, UInt16 *tempShiftP,
   Boolean *autoShiftedP)
```

Parameters

`capsLockP`
Returns `true` if caps lock is on.

`numLockP`
Returns `true` if num lock is on.

`tempShiftP`
Current temporary shift.

`autoShiftedP`
Returns `true` if shift is not set by the user but by the system, for example, at the beginning of a line.

Returns Always returns 0.

Compatibility Palm OS® 2.0 and later has more user-friendly auto shifting. It uses an upper case letter under these conditions:

- after an empty field

- after a period or other sentence terminator (such as ? or !).
- after two spaces

If the Graffiti 2 Feature Set is present and the Palm OS version is 4.1.2 or 5.2, the `capsLockP` and `numLockP` parameters are ignored. In later versions of Palm OS 4 beginning with 4.1.3, and later versions of Palm OS Garnet beginning with 5.2.1, all parameters work normally.

See Also GrfSetState()

GrfInitState Function

Purpose Reinitialize the Graffiti dictionary state.

Declared In `Graffiti.h`

Prototype

```
Err GrfInitState (void)
```

Parameters None.

Returns Always returns 0.

Compatibility If the Graffiti 2 Feature Set is present and the Palm OS version is 5.2, this function has no effect. It simply returns 0. In version 5.3SC, this function was reinstated.

See Also GrfGetState(), GrfSetState()

GrfMatch Function

Purpose Recognize the current stroke in the Graffiti point buffer and return with the recognized text.

Declared In `Graffiti.h`

Prototype

```
Err GrfMatch (UInt16 *flagsP, void *dataPtrP,
   UInt16 *dataLenP, UInt16 *uncertainLenP,
   GrfMatchInfoPtr matchInfoP)
```

Parameters

`flagsP`
: Glyph flags are returned here.

`dataPtrP`
: Return text is placed here.

`dataLenP`
: Size of `dataPtrP` on exit; number of characters returned on exit.

`uncertainLenP`
: Return number of uncertain characters.

`matchInfoP`
: Array of `grfMaxMatches`, or `NULL`.

Returns Returns 0 if no error, or `grfErrNoGlyphTable`, `grfErrNoDictionary`, or `grfErrNoMapping` if an error occurs.

Compatibility If the Graffiti 2 Feature Set is present, this function has no effect. It simply returns 0.

See Also GrfAddPoint(), GrfFlushPoints()

GrfMatchGlyph Function

Purpose Recognize the current stroke as a glyph.

Declared In `Graffiti.h`

Prototype

```
Err GrfMatchGlyph (GrfMatchInfoPtr matchInfoP,
    Int16 maxUnCertainty, UInt16 maxMatches)
```

Parameters

`matchInfoP`
: Pointer to array of matches to fill in.

`maxUnCertainty`
: Maximum number of errors to tolerate.

`maxMatches`
: Size of `matchInfoP` array.

Returns Returns 0 if no error, or `grfErrNoGlyphTable` if an error occurs.

Compatibility If the Graffiti 2 Feature Set is present, this function is deprecated and always returns `grfErrBadParam`.

See Also GrfMatch()

GrfProcessStroke Function

Purpose Translate a stroke to keyboard events using the Graffiti or Graffiti 2 engine.

Declared In `Graffiti.h`

Prototype

```
Err GrfProcessStroke (PointType *startPtP,
    PointType *endPtP, Boolean upShift)
```

Parameters

`startPtP`
: Start point of stroke.

`endPtP`
: End point of stroke.

`upShift`
: Set to `true` to feed an artificial upshift into the engine.

Returns Returns 0 if recognized.

Comments Called by `SysHandleEvent` when a `penUpEvent` is detected in the writing area. This routine recognizes the stroke and sends the recognized characters into the key queue. It also flushes the stroke out of the pen queue after recognition.

Compatibility If the Graffiti 2 Feature Set is present, the stroke is recognized using the Graffiti 2 engine rather than the Graffiti engine.

See Also `SysHandleEvent()`

GrfSetState Function

Purpose Set the current shift state.

Declared In `Graffiti.h`

Prototype

```
Err GrfSetState (Boolean capsLock,
    Boolean numLock, Boolean upperShift)
```

Parameters

`capsLock`
: Set to `true` to turn on caps lock.

`numLock`
: Set to `true` to turn on num lock.

`upperShift`
: Set to `true` to put into upper shift.

Returns Always returns 0.

Compatibility If the Graffiti 2 Feature Set is present and the Palm OS version is 4.1.2 or 5.2, the `capsLockP` and `numLockP` parameters are ignored. In later versions of Palm OS 4 beginning with 4.1.3, and later versions of Palm OS Garnet beginning with 5.2.1, all parameters work normally.

See Also `GrfGetState()`

39

Helper API

This chapter describes the Helper API declared in the header files `Helper.h` and `HelperServiceClass.h`. The Helper API is used when an application broadcasts a `sysNotifyHelperEvent` to all interested parties. The broadcaster of the notification and the notification clients (called **helpers**) use the Helper APIs to communicate with each other. The chapter discusses the following topics:

- Helper Data Structures
- Helper Constants

For more information on using the Helper API, see the section "Helper Notifications" on page 38 of the *Palm OS Programmer's Companion*, vol. I.

Helper Data Structures

HelperNotifyEnumerateListType Struct

Purpose The `HelperNotifyEnumerateListType` provides the broadcaster of the helper notification with information about the services that the helper can provide. This structure is used as the `data` field of the `HelperNotifyEventType` structure when the action code is `kHelperNotifyActionCodeEnumerate`.

Prototype

```
typedef struct HelperNotifyEnumerateListTypeTag {
  struct HelperNotifyEnumerateListTypeTag
    *nextP;
  Char helperAppName[kHelperAppMaxNameSize];
  Char actionName[kHelperAppMaxActionNameSize];
  UInt32 helperAppID;
  UInt32 serviceClassID;
} HelperNotifyEnumerateListType;
```

Note that the helper allocates this structure and then adds it to the linked list of structures pointed to by `notifyDetailsP->data.enumerateP` in the `SysNotifyParamType` that is sent to the helper. The helper should allocate one structure per supported service.

Even though the helper allocates this structure, the helper is not responsible for freeing the structure. Instead, the application that broadcast the notification must free the structure.

Fields

`nextP`
: A pointer to the next element in the list or `NULL` to signal the end of the list.

`helperAppName`
: A null-terminated string containing the name of the helper application, suitable for display in the user interface. If more than one application can perform the same service, this name is displayed as one of the choices in a pop-up list.

`actionName`
: A null-terminated string containing the name of the service that can be performed, suitable for display in the user interface. The action name should be short enough to display on a button.

`helperAppID`
: The helper's creator ID or any other ID that uniquely identifies the helper.

`serviceClassID`
: The ID of the service that the helper performs. See Helper Service Class IDs.

Compatibility

Implemented only if 4.0 New Feature Set is present.

HelperNotifyEventType Struct

Purpose

The `HelperNotifyEventType` structure contains all data associated with a helper notification (`sysNotifyHelperEvent`). A pointer to this structure is passed as the `notifyDetailsP` field in the `SysNotifyParamType` for that notification.

Prototype

```
typedef struct HelperNotifyEventTypeTag {
  UInt16 version;
  HelperNotifyActionCodeType actionCode;
  union {
    struct HelperNotifyEnumerateListTypeTag
      *enumerateP;
    struct HelperNotifyValidateTypeTag
      *validateP;
    struct HelperNotifyExecuteTypeTag
      *executeP;
  } data;
} HelperNotifyEventType;
```

Fields

`version`
: The version number for this structure. The current version is 1.

`actionCode`
: The action that the helper application should perform. See Table 39.1.

`data`
: Data specific to the action code. See Table 39.1.

The `HelperNotifyEventType` structure specifies which action is to be performed and contains data necessary for that action. All actions have some common data. Actions also have data specific to that action. The specific data uses a union that is part of the `HelperNotifyEventType` structure.

Table 39.1 Helper action codes

Action Code	data Type	Description
`kHelperNotifyActionCodeEnumerate`	`HelperNotifyEnumerateListType`	Send a list of available services.
`kHelperNotifyActionCodeValidate`	`HelperNotifyValidateType`	Validate the input data for the specified service.
`kHelperNotifyActionCodeExecute`	`HelperNotifyExecuteType`	Perform the specified service.

Compatibility Implemented only if 4.0 New Feature Set is present.

HelperNotifyExecuteType Struct

Purpose The `HelperNotifyExecuteType` structure identifies the service to perform and contains the data necessary to perform that service. This structure is used as the `data` field of the `HelperNotifyEventType` structure when the action code is `kHelperNotifyActionCodeExecute`.

Prototype

```
typedef struct HelperNotifyExecuteTypeTag {
  UInt32  serviceClassID;
  UInt32 helperAppID;
  Char *dataP;
  Char *displayedName;
  void *detailsP;
  Err err;
} HelperNotifyExecuteType;
```

Fields

`serviceClassID`
: The ID of the service to be performed. See Helper Service Class IDs.

`helperAppID`
: The unique ID of the helper; a value of 0 indicates that any available helper for the specified service class should perform the service.

`dataP`
: A null-terminated string specific to this service, such as a phone number for the dial service or an email address for the email service. See Table 39.2. Multiple fields must be separated by semicolons (;).

`displayedName`
: A null-terminated string containing an optional, human-readable description of the string in `dataP`. For example, if `dataP` contains a phone number, this field might contain the name of the person at that number.

`detailsP`
: A pointer to a data structure containing extra information that this service requires. See Table 39.2. If the service does not require extra information, this field is `NULL`.

`err`
: An error code that indicates if the service was performed successfully or not. If the service was successful, this field contains `errNone`, and the `handled` field in the notification data structure should be set to `true`.

The following table lists the Palm OS-defined values for the service class ID and for each service, shows what value `dataP` contains and what type of structure `detailsP` points to.

Table 39.2 HelperNotifyExecuteType values

Service Class ID	dataP Value	detailsP Value
`kHelperServiceClassIDVoiceDial`	The telephone number to dial	`NULL`
`kHelperServiceClassIDEMail`	The email address to which the message is to be sent	`HelperServiceEMailDetailsType`
`kHelperServiceClassIDSMS`	The SMS mailbox number to which the message is to be sent	`HelperServiceSMSDetailsType`
`kHelperServiceClassIDFax`	The fax number to which the fax is to be sent	`NULL`

Compatibility Implemented only if 4.0 New Feature Set is present.

HelperNotifyValidateType Struct

Purpose The `HelperNotifyValidateType` structure identifies a service that should be validated and the helper that should validate it. This structure is used as the `data` field of the `HelperNotifyEventType` structure when the action code is `kHelperNotifyActionCodeValidate`.

Prototype

```
typedef struct HelperNotifyValidateTypeTag {
  UInt32 serviceClassID;
  UInt32 helperAppID;
} HelperNotifyValidateType;
```

Fields

`serviceClassID`
: The ID of the service to be validated. See Helper Service Class IDs.

`helperAppID`
: The creator ID of the helper application. 0 indicates that any available helper for the specified service should respond. If nonzero, only the helper with the matching creator ID should respond.

The helper returns `true` in the `handled` field of the `SysNotifyParamType` structure to indicate that the service can be performed or `false` to indicate that the service cannot be performed.

Compatibility

Implemented only if 4.0 New Feature Set is present.

HelperServiceEMailDetailsType Struct

Purpose

The `HelperServiceEMailDetailsType` structure provides additional data for the email service. It is used as the `detailsP` field in the `HelperNotifyExecuteType` when the service class ID is `kHelperServiceClassIDEMail`.

Prototype

```
typedef struct _HelperServiceEMailDetailsType {
  UInt16 version;
  Char *cc;
  Char *subject;
  Char *message;
} HelperServiceEMailDetailsType;
```

Fields

`version`
: The version number for this structure. The current version is 1.

`cc`
: A null-terminated string containing an email address that should be sent a carbon copy of the message. Multiple addresses are separated by a semi-colon (;). May be `NULL` if there are no email addresses to carbon copy.

`subject`
: A null-terminated string containing the subject line. May be `NULL`.

`message`
: Initial message body string or `NULL`.

Compatibility Implemented only if 4.0 New Feature Set is present.

HelperServiceSMSDetailsType Struct

Purpose The `HelperServiceSMSDetailsType` structure provides the SMS message to be sent. It is used as the `detailsP` field in the `HelperNotifyExecuteType` when the service class ID is `kHelperServiceClassIDSMS`.

Prototype

```
typedef struct _HelperServiceSMSDetailsType {
  UInt16 version;
  Char *message;
} HelperServiceSMSDetailsType;
```

Fields

`version`
: The version number for this structure. The current version is 1.

`message`
: A null-terminated string containing the body of the message to be sent, or `NULL`.

Compatibility Implemented only if 4.0 New Feature Set is present.

Helper Constants

Helper Service Class IDs

The header file `HelperServiceClass.h` defines the constants listed in Table 39.3 as service class IDs. Third party developers may define their own service classes. To do so, you must register a 32-bit identifier with PalmSource, Inc. on this web site:

http://www.palmos.com/dev/creatorid/

Alternatively, you can use a creator ID that you already own.

Table 39.3 Service class ID constants

Constant	Value	Description
`kHelperServiceClassIDVoiceDial`	'voic'	Dial a phone number for a voice telephone call.
`kHelperServiceClassIDEMail`	'mail'	Send an email message.
`kHelperServiceClassIDSMS`	'sms_'	Send an SMS message.
`kHelperServiceClassIDFax`	'fax_'	Send a fax.

Compatibility Implemented only if 4.0 New Feature Set is present.

40

Key Manager

This chapter provides reference material for the key manager. The key manager API is declared in the header file `KeyMgr.h`.

For more information on the key manager, see "The Key Manager" on page 62 of the *Palm OS Programmer's Companion*, vol. I.

Key Manager Functions

KeyCurrentState Function

Purpose Return bit field with bits set for each key that is currently depressed.

Declared In `KeyMgr.h`

Prototype

```
UInt32 KeyCurrentState (void)
```

Parameters None.

Returns Returns a `UInt32` with bits set for keys that are depressed. See `keyBitPower`, `keyBitPageUp`, `keyBitPageDown`, etc., in `KeyMgr.h`.

Comments Called by applications that need to poll the keys.

See Also KeyRates()

KeyRates Function

Purpose Get or set the key repeat rates.

Declared In `KeyMgr.h`

Prototype

```
Err KeyRates (Boolean set, UInt16 *initDelayP,
   UInt16 *periodP, UInt16 *doubleTapDelayP,
   Boolean *queueAheadP)
```

Parameters

`set`
: If `true`, settings are changed; if `false`, current settings are returned.

`initDelayP`
: Initial delay in ticks for a auto-repeat event.

`periodP`
: Auto-repeat rate specified as period in ticks.

`doubleTapDelayP`
: Maximum double-tap delay, in ticks.

`queueAheadP`
: If `true`, auto-repeating keeps queueing up key events if the queue has keys in it. If `false`, auto-repeat doesn't enqueue keys unless the queue is already empty.

Returns Returns 0 if no error.

See Also KeyCurrentState()

KeySetMask Function

Purpose Specify which keys generate `keyDownEvents`.

You can specify this either by using this function or by using the `poweredOnKeyMask` modifier.

Declared In `KeyMgr.h`

Prototype

```
UInt32 KeySetMask (UInt32 keyMask)
```

Parameters

`keyMask`
: Mask with bits set for those keys to generate `keyDownEvents` for.

Returns Returns the old `key Mask`.

Compatibility Implemented only if 2.0 New Feature Set is present.

41

Locale Manager

This chapter describes the Locale Manager API as described in the header files `LocaleMgr.h`, `Localize.h`, and `PalmLocale.h`. It discusses the following topics:

- Locale Manager Data Types
- Locale Manager Constants
- Locale Manager Functions

For more information on the Locale Manager, see the chapter "Localized Applications" on page 385 of the *Palm OS Programmer's Companion*, vol. I.

Locale Manager Data Types

CountryType Typedef

Purpose The `CountryType` defines a country code. The Country Constants in `PalmLocale.h` define the possible values for `CountryType` variables.

Prototype

```
typedef UInt8 CountryType;
```

Compatibility Prior to version 4.0, `CountryType` was an enum in `Preferences.h` that defined only 33 country codes. The Palm OS® 4.0 definition of `CountryType` is compatible with the previous definition.

LanguageType Typedef

Purpose The `LanguageType` defines a language code. The Language Constants in `PalmLocale.h` define the possible values for `LanguageType` variables.

Prototype

```
typedef UInt8 LanguageType;
```

Compatibility

The `LanguageType` definition was added in Palm OS 3.5. Prior to version 4.0, `LanguageType` was an enum in `Preferences.h` that defined only eight language codes. The Palm OS 4.0 definition of `LanguageType` is compatible with the previous definition.

LmLocaleType Struct

Purpose

The `LmLocaleType` struct defines the country and language used in a locale.

Prototype

```
struct _LmLocaleType {
  UInt16 language;
  UInt16 country;
};
typedef struct _LmLocaleType LmLocaleType;
```

Fields

`language`
: One of the Language Constants. This value identifies the language spoken in the current locale.

`country`
: One of the Country Constants. This value identifies the locale's country, which helps to identify the language dialect. For example, a language of `lEnglish` specifies a different dialect if the country is `cUnitedKingdom` than if it is `cUnitedStates`.

Note that the `language` and `country` fields are type `UInt16` instead of `LanguageType` and `CountryType`.

Compatibility

The `LmLocaleType` is defined only if 4.0 New Feature Set is present. It supersedes the `OmLocaleType` that was introduced with Palm OS 3.5. `LmLocaleType` is bit-compatible with `OmLocaleType`.

NumberFormatType Enum

Purpose

The `NumberFormatType` enum specifies how numbers are formatted. You can pass a `NumberFormatType` value to `LocGetNumberSeparators()` and receive the appropriate separator characters for thousands and decimals.

Prototype

```
typedef enum {
  nfCommaPeriod,
  nfPeriodComma,
  nfSpaceComma,
  nfApostrophePeriod,
  nfApostropheComma
} NumberFormatType;
```

Constants

`nfCommaPeriod`
: Uses a comma (,) as the thousands separator and a period (.) as the decimal separator.

`nfPeriodComma`
: Uses a period as the thousands separator and a comma as the decimal separator.

`nfSpaceComma`
: Uses a space ( ) as the thousands separator and a comma as the decimal separator.

`nfApostrophePeriod`
: Uses an apostrophe (') as the thousands separator and a period as the decimal separator.

`nfApostropheComma`
: Uses an apostrophe as the thousands separator and a comma as the decimal separator.

Locale Manager Constants

Character Encoding Constants

The `PalmLocale.h` file defines several character encoding constants that are used as values of `CharEncodingType` variables. The character encoding constants generally follow the format:

```
charEncoding Name
```

where *Name* is the name of the character encoding.

The following table shows examples of the character encoding constants. For a complete list, see the `PalmLocale.h` file.

Constant	Description
`charEncodingUnknown`	Unknown to this version of Palm OS®
`charEncodingAscii`	ISO 646-1991
`charEncodingISO8859_1`	ISO 8859 Part 1 (also known as ISO Latin 1). This encoding is commonly used for the Roman alphabet
`charEncodingPalmLatin`	Palm OS version of Microsoft Windows code page 1252. This encoding is identical to code page 1252 with Palm-specific characters added in the control range.
`charEncodingShiftJIS`	Encoding for 0208-1990 with single-byte Japanese Katakana. This encoding is commonly used for Japanese alphabets.
`charEncodingPalmSJIS`	Palm OS version of Microsoft Windows code page 932. This encoding is identical to code page 932, with Palm-specific characters added in the control range and with a Yen symbol instead of the Reverse Solidus at location 0x5c.
`charEncodingCP1252`	Microsoft Windows extensions to ISO 8859 Part 1
`charEncodingCP932`	Microsoft Windows extensions to Shift JIS
`charEncodingUTF8`	Eight-bit safe encoding for Unicode

Country Constants

The `PalmLocale.h` file defines several country constants that are used as values of `CountryType` variables. The country type constants have the following format:

`c`*`CountryName`*

where *CountryName* is the name of the country. There is one constant for each country identified in the ISO 3166 standard, which currently defines 239 countries.

The following table shows examples of the country type constants. For a complete list, see the `PalmLocale.h` file.

Constant	Description
`cAustralia`	Australia
`cAustria`	Austria
`cBelgium`	Belgium

Language Constants

The `PalmLocale.h` file defines several language constants that are used as values of `LanguageType` variables. The language type constants have the following format:

`l`*`LanguageName`*

where *LanguageName* is the name of the language. There is one constant for each language specified in the ISO 639 standard, which currently defines 137 languages.

The following table shows examples of the language type constants. For a complete list, see the `PalmLocale.h` file.

Constant	Description
`lEnglish`	English
`lFrench`	French
`lGerman`	German

Locale Manager Size Constants

You can use the Locale Manager size constants to determine the size of strings to allocate for some of the locale settings.

NOTE: The constants in the table below do not count the terminating null character. Therefore, you need to allocate a string of size `kMaxCountryNameLen+1` to hold a country name, for example.

Constant	Value	Description
`kMaxCountryNameLen`	19	The maximum length of a country name string.
`kMaxCurrencyNameLen`	19	The maximum length of a currency name string.
`kMaxCurrencySymbolLen`	5	The maximum length of a currency symbol string.

Locale Manager Functions

LmGetLocaleSetting Function

Purpose Return the requested setting for a given locale.

Declared In `LocaleMgr.h`

Prototype

```
Err LmGetLocaleSetting (UInt16 iLocaleIndex,
   LmLocaleSettingChoice iChoice, void *oValue,
   UInt16 iValueSize)
```

Parameters

→ *`iLocaleIndex`*
Index of the locale whose settings you want to retrieve.

→ *`iChoice`*
The setting you want to retrieve. This is a constant in the form `lmChoice...`; see Table 41.1 for a list of possible values.

← *oValue*
The value of the `iChoice` setting. The size of this buffer depends on the value of `iChoice`, as shown in Table 41.1.

→ *iValueSize*
The size of the `oValue` buffer.

Returns Returns one of the following values:

`errNone`
Success.

`lmErrBadLocaleIndex`
`iLocaleIndex` is out of range.

`lmErrSettingDataOverflow`
The `oValue` buffer is too small to hold the setting's value.

`lmErrBadLocaleSettingChoice`
The `iChoice` parameter contains an unknown or unsupported value.

Comments This function accesses the private locale system resource and returns the requested information in the `oValue` parameter. The size and type of the `oValue` parameter depend on which setting you want to retrieve. Table 41.1 lists and describes the possible settings and the type of data returned in `oValue` for each setting. For fixed-size values, make sure that the size of the `oValue` buffer is exactly the size of the returned value. It should be neither larger than nor smaller than the size of the returned value.

This function returns the default settings for the locale. Users can override many of the locale settings using the Preferences application. Applications should always honor the user's preferences rather than the locale defaults. For this reason, it's recommended that if a corresponding system preference is available, you should check it instead (using `PrefGetPreference()`). Use `LmGetLocaleSetting` only if you want to retrieve values that the user cannot override (such as the country name or currency symbol) or if you want to retrieve information about a locale other than the current locale.

Table 41.1 LmGetLocaleSetting choices and sizes

lmChoice...	oValue Data Type	Description
`CountryName`	String buffer of size `kMaxCountryNameLen+1` bytes	The name of the locale's country.
`CurrencyName`	String buffer of size `kMaxCurrencyNameLen+1` bytes	The name of the currency used in this locale.
`CurrencySymbol`	String buffer of size `kMaxCurrencySymbolLen+1` bytes	The symbol used to denote monetary values in this locale.
`CurrencyDecimalPlaces`	`UInt16`	The number of decimal places that monetary values are typically given.
`DateFormat`	`DateFormatType`	The short date format used in this locale. For example: 95/12/31
`InboundDefaultVObjectEncoding`	`CharEncodingType`	Inbound encoding for vObjects with no CHARSET property.
`Locale`	`LmLocaleType`	A structure containing the locale's language and country codes.
`LongDateFormat`	`DateFormatType`	The long date format used in this locale. For example: 31 Dec 1995
`MeasurementSystem`	`MeasurementSystemType`	The measurement system (metric system or English system) used in this locale.

Table 41.1 LmGetLocaleSetting choices and sizes *(continued)*

lmChoice...	oValue Data Type	Description
NumberFormat	NumberFormatType	The format used for numbers, with regards to the thousands separator and the decimal point, in this locale.
OutboundVObject Encoding	CharEncodingType	Outbound encoding for vObjects.
PrimaryEmail Encoding	CharEncodingType	First attempt at Email encoding.
PrimarySMS Encoding	CharEncodingType	First attempt at SMS encoding.
SecondaryEmail Encoding	CharEncodingType	Second attempt at Email encoding.
SecondarySMS Encoding	CharEncodingType	Second attempt at SMS encoding.
SupportsLunar Calendar	Boolean	Does locale support Chinese Lunar Calendar?
TimeFormat	TimeFormatType	The format used for time values in this locale.
TimeZone	Int16	The locale's default time zone given as the number of minutes east of Greenwich Mean Time (GMT).

Table 41.1 LmGetLocaleSetting choices and sizes *(continued)*

lmChoice...	oValue Data Type	Description
`UniqueCurrencySymbol`	String buffer of size `kMaxCurrencySymbolLen+1` bytes	A unique symbol for monetary values. For example, the symbol `$` is used both for US dollars and Portuguese escudos. The unique currency symbol for US dollars is `US$`.
`WeekStartDay`	`UInt16`	The first day of the week (Sunday or Monday) in this locale. Days of the week are numbered from 0 to 6 starting with Sunday = 0.

Compatibility Implemented only if 4.0 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `LmGlueGetLocaleSetting`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `LmGetNumLocales()`, `LmLocaleToIndex()`

LmGetNumLocales Function

Purpose Return the number of known locales.

Declared In `LocaleMgr.h`

Prototype

```
UInt16 LmGetNumLocales (void)
```

Parameters `None.`

Returns Returns the number of locales that the locale system resource defines.

Comments Use this function to obtain the range of possible values that you can pass as an index to `LmGetLocaleSetting()`. If `LmGetNumLocales` returns 3, then `LmGetLocaleSetting` accepts indexes in the range of 0 to 2.

This function returns only the number of locales for which the ROM has locale information. It does not return the number of locales that could possibly be defined. For example, the system resource currently contains no locale whose language is `lHebrew` and country is `cIsrael`, even though that is a valid locale.

Compatibility Implemented only if 4.0 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `LmGlueGetNumLocales`. For more information, see Chapter 80, "PalmOSGlue Library."

LmLocaleToIndex Function

Purpose Convert an `LmLocaleType` to an index.

Declared In `LocaleMgr.h`

Prototype

```
Err LmLocaleToIndex (const LmLocaleType *iLocale,
    UInt16 *oLocaleIndex)
```

Parameters

→ *iLocale*
The locale to convert.

← *oLocaleIndex*
The index of `iLocale` upon return.

Returns Returns `errNone` upon success or `lmErrUnknownLocale` if the locale could not be found.

Comments You can use this function to obtain a valid index to pass to `LmGetLocaleSetting()`. For example, you might use the Overlay Manager routine `OmGetSystemLocale()` to return the locale used on the current system and then pass that locale to this function to obtain its index.

```
LmLocaleType locale;
Char oValue[kMaxCurrencySymbolLen+1];
UInt16 index;
```

```
OmGetSystemLocale(&locale);
LmLocaleToIndex(&locale, &index);
LmGetLocaleSetting(index,
  lmChoiceCurrencySymbol, oValue,
  sizeof(oValue));
```

The `LmLocaleType` that is passed in `iLocale` can use the constants `lmAnyCountry` or `lmAnyLanguage` as wildcards. For example, if the country is `lmAnyCountry`, `LmLocaleToIndex` returns the index of the first locale that matches the language.

Compatibility

Implemented only if 4.0 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `LmGlueLocaleToIndex`. For more information, see Chapter 80, "PalmOSGlue Library."

42

Memory Manager

This chapter provides reference information for the memory manager. The memory manager API is declared in the header file `MemoryMgr.h`.

For more information on the memory manager, see the chapter "Memory" in the *Palm OS Programmer's Companion*, vol. I.

Memory Manager Functions

MemCardInfo Function

Purpose Return information about a memory card.

Declared In `MemoryMgr.h`

Prototype

```
Err MemCardInfo (UInt16 cardNo, Char *cardNameP,
   Char *manufNameP, UInt16 *versionP,
   UInt32 *crDateP, UInt32 *romSizeP,
   UInt32 *ramSizeP, UInt32 *freeBytesP)
```

Parameters

`cardNo`
: Card number.

`cardNameP`
: Pointer to character array (32 bytes), or 0.

`manufNameP`
: Pointer to character array (32 bytes), or 0.

`versionP`
: Pointer to version variable, or 0.

`crDateP`
: Pointer to creation date variable, or 0.

`romSizeP`
: Pointer to ROM size variable, or 0.

`ramSizeP`
Pointer to RAM size variable, or 0.

`freeBytesP`
Pointer to free byte-count variable, or 0.

Returns Returns 0 if no error.

Comments Pass 0 for those variables that you don't want returned.

MemCmp Function

Purpose Compare two blocks of memory.

NOTE: Blocks are compared as unsigned bytes.

Declared In `MemoryMgr.h`

Prototype

```
Int16 MemCmp (const void *s1, const void *s2,
    Int32 numBytes)
```

Parameters

`s1, s2`
Pointers to block of memory.

`numBytes`
Number of bytes to compare.

Returns Zero if they match, non-zero if not:

+ if `s1` > `s2`

- if `s1` < `s2`

Compatibility Implemented only if 2.0 New Feature Set is present.

`MemCmp` can be used to test the equality of blocks in memory on all versions that support `MemCmp`; however, testing the sort ordering of blocks in memory works reliably only on Palm OS® versions 3.5 and higher. On versions earlier than 3.2, `MemCmp` always returns a positive value if the blocks are unequal. On versions 3.2 and 3.3, `MemCmp` reliably returns positive to indicate `s1` > `s2` and negative to indicate `s1` < `s2` *only* if the characters that differ are less than 128 apart. If the difference is greater than that, `MemCmp` may return positive when it should return negative and vice versa.

MemDebugMode Function

Purpose Return the current debugging mode of the memory manager.

Declared In `MemoryMgr.h`

Prototype
```
UInt16 MemDebugMode (void)
```

Parameters No parameters.

Returns Returns debug flags as described for `MemSetDebugMode()`.

MemHandleCardNo Function

Purpose Return the card number a chunk resides in.

Declared In `MemoryMgr.h`

Prototype
```
UInt16 MemHandleCardNo (MemHandle h)
```

Parameters → *h*
Chunk handle.

Returns Returns the card number.

Comments Call this routine to retrieve the card number (0 or 1) a movable chunk resides on.

See Also `MemPtrCardNo()`

MemHandleDataStorage Function

Purpose Return `true` if the given handle is part of a data storage heap. If not, it's a handle in the dynamic heap.

Declared In `MemoryMgr.h`

Prototype
```
Boolean MemHandleDataStorage (MemHandle h)
```

Parameters → *h*
Chunk handle.

Returns Returns `true` if the handle is part of a data storage heap.

Comments Called by Fields package routines to determine if they need to worry about data storage write-protection when editing a text field.

See Also `MemPtrDataStorage()`

MemHandleFree Function

Purpose Dispose of a movable chunk.

Declared In `MemoryMgr.h`

Prototype
```
Err MemHandleFree (MemHandle h)
```

Parameters → *h*
Chunk handle.

Returns Returns 0 if no error, or `memErrInvalidParam` if an error occurs.

Comments Call this routine to dispose of a movable chunk.

See Also `MemHandleNew()`

MemHandleHeapID Function

Purpose Return the heap ID of a chunk.

Declared In `MemoryMgr.h`

Prototype
```
UInt16 MemHandleHeapID (MemHandle h)
```

Parameters → *h*
Chunk handle.

Returns Returns the heap ID of a chunk.

Comments Call this routine to get the heap ID of the heap a chunk resides in.

See Also `MemPtrHeapID()`

MemHandleLock Function

Purpose Lock a chunk and obtain a pointer to the chunk's data.

Declared In `MemoryMgr.h`

Prototype
```
MemPtr MemHandleLock (MemHandle h)
```

Parameters → *h*
Chunk handle.

Returns Returns a pointer to the chunk.

Comments Call this routine to lock a chunk and obtain a pointer to the chunk.

`MemHandleLock` and `MemHandleUnlock` should be used in pairs.

See Also `MemHandleNew()`, `MemHandleUnlock()`

MemHandleNew Function

Purpose Allocate a new movable chunk in the dynamic heap and returns a handle to it.

Declared In `MemoryMgr.h`

Prototype

```
MemHandle MemHandleNew (UInt32 size)
```

Parameters

→ *size*
The desired size of the chunk.

Returns Returns a handle to the new chunk, or 0 if unsuccessful.

Comments Use this call to allocate dynamic memory. Before you can write data to the memory chunk that `MemHandleNew` allocates, you must call `MemHandleLock()` to lock the chunk and get a pointer to it.

See Also `MemPtrFree()`, `MemPtrNew()`, `MemHandleFree()`, `MemHandleLock()`

MemHandleResize Function

Purpose Resize a chunk.

Declared In `MemoryMgr.h`

Prototype

```
Err MemHandleResize (MemHandle h, UInt32 newSize)
```

Parameters

→ *h*
Chunk handle.

→ *newSize*
The new desired size.

Returns

`0`
No error.

`memErrInvalidParam`
Invalid parameter passed.

`memErrNotEnoughSpace`
Not enough free space in heap to grow chunk.

`memErrChunkLocked`
Can't grow chunk because it's locked.

Comments Call this routine to resize a chunk. This routine is always successful when shrinking the size of a chunk, even if the chunk is locked. When growing a chunk, it first attempts to grab free space immediately following the chunk so that the chunk does not have to move. If the chunk has to move to another free area of the heap to grow, it must be movable and have a lock count of 0.

On devices running version 2.0 or earlier of Palm OS, the `MemHandleResize` function tries to resize the chunk only within the same heap, whereas `DmResizeRecord()` will look for space in other data heaps if it can't find enough space in the original heap.

See Also `MemHandleNew()`, `MemHandleSize()`

MemHandleSetOwner Function

Purpose Set the owner ID of a chunk.

Declared In `MemoryMgr.h`

Prototype

```
Err MemHandleSetOwner (MemHandle h, UInt16 owner)
```

Parameters

→ *h*
Chunk handle.

→ *owner*
New owner ID of the chunk. Specify 0 to set the owner to the operating system.

Returns Returns 0 if no error, or `memErrInvalidParam` if an error occurs.

Comments When you allocate a parameter block to pass to `SysUIAppSwitch()` or `SysAppLaunch()`, you must call `MemPtrSetOwner()` to grant ownership of the parameter block chunk to the OS (your application is originally set as the owner). If the parameter block structure references any chunks by handle, you also must call `MemHandleSetOwner` to grant ownership of those blocks to the OS. If you don't change the ownership of these chunks, they will get freed before the application you're launching has a chance to use them.

IMPORTANT: Once you have granted ownership of a memory chunk to the system, do not attempt to free it yourself. The operating system will free it when the application invoked with `SysUIAppSwitch()` or `SysAppLaunch()` quits.

MemHandleSize Function

Purpose Return the requested size of a chunk.

Declared In `MemoryMgr.h`

Prototype
```
UInt32 MemHandleSize (MemHandle h)
```

Parameters → *h*
Chunk handle.

Returns Returns the requested size of the chunk.

Comments Call this routine to get the size originally requested for a chunk.

See Also `MemHandleResize()`

MemHandleToLocalID Function

Purpose Convert a handle into a local chunk ID which is card relative.

Declared In `MemoryMgr.h`

Prototype
```
LocalID MemHandleToLocalID (MemHandle h)
```

Parameters → *h*
Chunk handle.

Returns Returns local ID, or NULL (0) if unsuccessful.

Comments Call this routine to convert a chunk handle to a local ID.

See Also `MemLocalIDToGlobal()`, `MemLocalIDToLockedPtr()`

MemHandleUnlock Function

Purpose Unlock a chunk given a chunk handle.

Declared In `MemoryMgr.h`

Prototype
```
Err MemHandleUnlock (MemHandle h)
```

Parameters → *h*
The chunk handle.

Returns `0`
No error.

`memErrInvalidParam`
Invalid parameter passed.

Comments Call this routine to decrement the lock count for a chunk.

`MemHandleLock` and `MemHandleUnlock` should be used in pairs.

See Also `MemHandleLock()`

MemHeapCheck Function

Purpose Check validity of a given heap.

Declared In `MemoryMgr.h`

Prototype
```
Err MemHeapCheck (UInt16 heapID)
```

Parameters `heapID`
ID of heap to check.

Returns Returns 0 if no error.

See Also `MemDebugMode()`, `MemSetDebugMode()`

MemHeapCompact Function

Purpose Compact a heap.

Declared In `MemoryMgr.h`

Prototype
```
Err MemHeapCompact (UInt16 heapID)
```

Parameters → *heapID*
ID of the heap to compact.

Returns Always returns 0.

Comments Most applications never need to call this function explicitly. The system software calls this function at various times; for example, during memory allocation (if sufficient free space is not available) and during system reboot.

Call this routine to compact a heap and merge all free space. This routine attempts to move all movable chunks to the start of the heap and merge all free space in the center of the heap.

MemHeapDynamic Function

Purpose Return `true` if the given heap is a dynamic heap.

Declared In `MemoryMgr.h`

Prototype

```
Boolean MemHeapDynamic (UInt16 heapID)
```

Parameters

`heapID`
ID of the heap to be tested.

Returns Returns `true` if dynamic, `false` if not.

Comments Dynamic heaps are used for volatile storage, application stacks, globals, and dynamically allocated memory.

> **NOTE:** In Palm OS 3.5, the dynamic heap is sized based on the amount of memory available, and is generally larger than before.

See Also `MemNumHeaps()`, `MemHeapID()`

MemHeapFlags Function

Purpose Return the heap flags for a heap.

Declared In `MemoryMgr.h`

Prototype

```
UInt16 MemHeapFlags (UInt16 heapID)
```

Parameters

→ *`heapID`*
ID of heap.

Returns Returns the heap flags.

Comments Call this routine to retrieve the heap flags for a heap. The flags can be examined to determine if the heap is ROM based or not. ROM-based heaps have the `memHeapFlagReadOnly` bit set (the `memHeapFlagReadOnly` mask has a value of `0x0001`).

See Also `MemNumHeaps()`, `MemHeapID()`

MemHeapFreeBytes Function

Purpose Return the total number of free bytes in a heap and the size of the largest free chunk in the heap.

Declared In `MemoryMgr.h`

Prototype

```
Err MemHeapFreeBytes (UInt16 heapID,
   UInt32 *freeP, UInt32 *maxP)
```

Parameters

→ *heapID*
ID of heap.

↔ *freeP*
Pointer to a variable of type `UInt32` for free bytes.

↔ *maxP*
Pointer to a variable of type `UInt32` for max free chunk size. Do not pass `NULL` for this argument.

Returns Always returns 0.

Comments This routine doesn't compact the heap but may be used to determine in advance whether an allocation request will succeed. Before allocating memory, call this function and test the value returned in maxP to determine whether enough free space to fulfill your allocation request exists. If not, you may make more space available by calling the `MemHeapCompact()` function. Note that both `MemPtrNew()` and `MemHandleNew()` automatically compact the heap if necessary.

See Also `MemHeapSize()`, `MemHeapID()`, `MemHeapCompact()`

MemHeapID Function

Purpose Return the heap ID for a heap, given its index and the card number.

Declared In `MemoryMgr.h`

Prototype

```
UInt16 MemHeapID (UInt16 cardNo,
   UInt16 heapIndex)
```

Parameters

→ *cardNo*
The card number, either 0 or 1.

→ *heapIndex*
The heap index, anywhere from 0 to `MemNumHeaps()` - 1.

Returns Returns the heap ID.

Comments Call this routine to retrieve the heap ID of a heap, given the heap index and the card number. A heap ID must be used to obtain information on a heap such as its size, free bytes, etc., and is also passed to any routines which manipulate heaps.

See Also `MemNumHeaps()`

MemHeapScramble Function

Purpose Scramble the specified heap.

Declared In `MemoryMgr.h`

Prototype

```
Err MemHeapScramble (UInt16 heapID)
```

Parameters

`heapID`
ID of heap to scramble.

Comments The system attempts to move each movable chunk.

Useful for debugging.

Returns Always returns 0.

See Also `MemDebugMode()`, `MemSetDebugMode()`

MemHeapSize Function

Purpose Return the total size of a heap including the heap header.

Declared In `MemoryMgr.h`

Prototype
```
UInt32 MemHeapSize (UInt16 heapID)
```

Parameters → *heapID*
ID of heap.

Returns Returns the total size of the heap.

See Also `MemHeapFreeBytes()`, `MemHeapID()`

MemLocalIDKind Function

Purpose Return whether or not a local ID references a handle or a pointer.

Declared In `MemoryMgr.h`

Prototype
```
LocalIDKind MemLocalIDKind (LocalID local)
```

Parameters → *local*
Local ID to query

Returns Returns `LocalIDKind`, or a `memIDHandle` or `memIDPtr` (see `MemoryMgr.h`).

Comments This routine determines if the given local ID is to a nonmovable (`memIDPtr`) or movable (`memIDHandle`) chunk.

MemLocalIDToGlobal Function

Purpose Convert a local ID, which is card relative, into a global pointer in the designated card.

Declared In `MemoryMgr.h`

Prototype
```
MemPtr MemLocalIDToGlobal (LocalID local,
   UInt16 cardNo)
```

Parameters → *local*
The local ID to convert.

→ *cardNo*
Memory card the chunk resides in.

Returns Returns pointer or handle to chunk.

See Also MemLocalIDKind(), MemLocalIDToLockedPtr()

MemLocalIDToLockedPtr Function

Purpose Return a pointer to a chunk given its local ID and card number.

If the local ID references a movable chunk handle, this routine automatically locks the chunk before returning.

Declared In MemoryMgr.h

Prototype

```
MemPtr MemLocalIDToLockedPtr (LocalID local,
    UInt16 cardNo)
```

Parameters

`local`
Local chunk ID.

`cardNo`
Card number.

Returns Returns pointer to chunk, or 0 if an error occurs.

See Also MemLocalIDToGlobal(), MemLocalIDToPtr(), MemLocalIDKind(), MemPtrToLocalID(), MemHandleToLocalID()

MemLocalIDToPtr Function

Purpose Return pointer to chunk, given the local ID and card number.

Declared In MemoryMgr.h

Prototype

```
MemPtr MemLocalIDToPtr (LocalID local,
    UInt16 cardNo)
```

Parameters

→ *local*
Local ID to query.

→ *cardNo*
Card number the chunk resides in.

Returns Returns a pointer to the chunk, or 0 if error.

Comments If the local ID references a movable chunk and that chunk is **not** locked, this function returns 0 to indicate an error.

See Also `MemLocalIDToGlobal()`, `MemLocalIDToLockedPtr()`

MemMove Function

Purpose Copy a number of bytes from one location to another in the dynamic heap.

Declared In `MemoryMgr.h`

Prototype

```
Err MemMove (void *destP, const void *sourceP,
    Int32 numBytes)
```

Parameters

`destP`
Pointer to destination.

`sourceP`
Pointer to source.

`numBytes`
Number of bytes to move.

Returns Always returns 0.

Comments Like the ISO C libary call `memmove`, this function copies *numBytes* bytes from the object pointed to by *sourceP* into the object pointed to by *destP*. The source and destination may overlap; the copy is always done in a non-destructive manner.

For operations where the destination is in a data heap, see `DmSet()`, `DmWrite()`, and related functions.

MemNumCards Function

Purpose Return the number of memory card slots in the system. Not all slots need to be populated.

Declared In `MemoryMgr.h`

Prototype

```
UInt16 MemNumCards (void)
```

Parameters None.

Returns Returns number of slots in the system.

MemNumHeaps Function

Purpose Return the number of heaps available on a particular card.

Declared In `MemoryMgr.h`

Prototype
```
UInt16 MemNumHeaps (UInt16 cardNo)
```

Parameters → *cardNo*
The card number; either 0 or 1.

Returns Number of heaps available, including ROM- and RAM-based heaps.

Comments Call this routine to retrieve the total number of heaps on a memory card. The information can be obtained by calling `MemHeapSize()`, `MemHeapFreeBytes()`, and `MemHeapFlags()` on each heap using its heap ID. The heap ID is obtained by calling `MemHeapID()` with the card number and the heap index, which can be any value from 0 to `MemNumHeaps`.

MemNumRAMHeaps Function

Purpose Return the number of RAM heaps in the given card.

Declared In `MemoryMgr.h`

Prototype
```
UInt16 MemNumRAMHeaps (UInt16 cardNo)
```

Parameters `cardNo`
The card number.

Returns Returns the number of RAM heaps.

See Also `MemNumCards()`

MemPtrCardNo Function

Purpose Return the card number (0 or 1) a nonmovable chunk resides on.

Declared In `MemoryMgr.h`

Prototype
```
UInt16 MemPtrCardNo (MemPtr p)
```

Parameters → *p*
Pointer to the chunk.

Returns Returns the card number.

See Also `MemHandleCardNo()`

MemPtrDataStorage Function

Purpose Return `true` if the given pointer is part of a data storage heap; if not, it is a pointer in the dynamic heap.

Declared In `MemoryMgr.h`

Prototype

```
Boolean MemPtrDataStorage (MemPtr p)
```

Parameters

`p`
Pointer to a chunk.

Returns Returns `true` if the chunk is part of a data storage heap.

Comments Called by Fields package to determine if it needs to worry about data storage write-protection when editing a text field.

See Also `MemHeapDynamic()`

MemPtrFree Macro

Purpose Dispose of a chunk.

Declared In `MemoryMgr.h`

Prototype

```
#define MemPtrFree (p)
```

Parameters

→ *p*
Pointer to a chunk.

Returns `0`
If no error or `memErrInvalidParam` (invalid parameter).

Comments Call this routine to dispose of a nonmovable chunk.

MemPtrHeapID Function

Purpose Return the heap ID of a chunk.

Declared In `MemoryMgr.h`

Prototype
```
UInt16 MemPtrHeapID (MemPtr p)
```

Parameters → *p*
Pointer to the chunk.

Returns Returns the heap ID of a chunk.

Comments Call this routine to get the heap ID of the heap a chunk resides in.

MemPtrNew Function

Purpose Allocate a new nonmovable chunk in the dynamic heap.

Declared In `MemoryMgr.h`

Prototype
```
MemPtr MemPtrNew (UInt32 size)
```

Parameters → *size*
The desired size of the chunk.

Returns Returns pointer to the new chunk, or 0 if unsuccessful.

Comments This routine allocates a nonmovable chunk in the dynamic heap and returns a pointer to the chunk. Applications can use it when allocating dynamic memory. Note that chunks range in size from 1 byte to slightly less than 64KB; you cannot allocate a single chunk larger than this.

In Palm OS 3.5, the dynamic heap is sized based on the amount of memory available, and is generally larger than before.

NOTE: You cannot allocate a zero-size reference block.

See Also `MemGluePtrNew()`

MemPtrRecoverHandle Function

Purpose Recover the handle of a movable chunk, given a pointer to its data.

Declared In `MemoryMgr.h`

Prototype

```
MemHandle MemPtrRecoverHandle (MemPtr p)
```

Parameters

→ *p*
Pointer to the chunk.

Returns Returns the handle of the chunk, or 0 if unsuccessful.

Comments Don't call this function for pointers in ROM or nonmovable data chunks.

MemPtrResize Function

Purpose Resize a chunk.

Declared In `MemoryMgr.h`

Prototype

```
Err MemPtrResize (MemPtr p, UInt32 newSize)
```

Parameters

→ *p*
Pointer to the chunk.

→ *newSize*
The new desired size.

Returns Returns 0 if no error, or `memErrNotEnoughSpace` `memErrInvalidParam`, or `memErrChunkLocked` if an error occurs.

Comments Call this routine to resize a locked chunk. This routine is always successful when shrinking the size of a chunk. When growing a chunk, it attempts to use free space immediately following the chunk.

See Also `MemPtrSize()`, `MemHandleResize()`

MemPtrSetOwner Function

Purpose Set the owner ID of a chunk.

Declared In `MemoryMgr.h`

Prototype

```
Err MemPtrSetOwner (MemPtr p, UInt16 owner)
```

Parameters

→ *p*
Pointer to the chunk.

→ *owner*
New owner ID of the chunk. Specify 0 to set the owner to the operating system.

Returns Returns 0 if no error, or `memErrInvalidParam` if an error occurs.

Comments When you allocate a parameter block to pass to `SysUIAppSwitch()` or `SysAppLaunch()`, you must call `MemPtrSetOwner()` or `MemHandleSetOwner()` to grant ownership of the parameter block chunk, and any other chunks referenced in it, to the OS (your application is originally set as the owner). If you don't change the ownership of the parameter block, it will get freed before the application you're launching has a chance to use it.

IMPORTANT: Once you have granted ownership of a memory chunk to the system, do not attempt to free it yourself. The operating system will free it when the application invoked with `SysUIAppSwitch()` or `SysAppLaunch()` quits.

MemPtrSize Function

Purpose Return the size of a chunk.

Declared In `MemoryMgr.h`

Prototype

```
UInt32 MemPtrSize (MemPtr p)
```

Parameters

→ *p*
Pointer to the chunk.

Returns The requested size of the chunk.

Comments Call this routine to get the original requested size of a chunk.

MemPtrToLocalID Function

Purpose Convert a pointer into a card-relative local chunk ID.

Declared In `MemoryMgr.h`

Prototype
```
LocalID MemPtrToLocalID (MemPtr p)
```

Parameters → *p*
Pointer to a chunk.

Returns Returns the local ID of the chunk.

Comments Call this routine to convert a chunk pointer to a local ID.

See Also `MemLocalIDToPtr()`

MemPtrUnlock Function

Purpose Unlock a chunk, given a pointer to the chunk.

Declared In `MemoryMgr.h`

Prototype
```
Err MemPtrUnlock (MemPtr p)
```

Parameters `p`
Pointer to a chunk.

Returns 0 if no error, or `memErrInvalidParam` if an error occurs.

Comments A chunk must **not** be unlocked more times than it was locked.

See Also `MemHandleLock()`

MemSet Function

Purpose Set a memory range in a dynamic heap to a specific value.

Declared In `MemoryMgr.h`

Prototype
```
Err MemSet (void *dstP, Int32 numBytes,
   UInt8 value)
```

Parameters `dstP`
Pointer to the destination.

`numBytes`
Number of bytes to set.

`value`
Value to set.

Returns Always returns 0.

Comments For operations where the destination is in a data heap, see `DmSet()`, `DmWrite()`, and related functions.

MemSetDebugMode Function

Purpose Set the debugging mode of the memory manager.

Declared In `MemoryMgr.h`

Prototype

```
Err MemSetDebugMode (UInt16 flags)
```

Parameters `flags`
Debug flags.

Comments Use the logical OR operator (|) to provide any combination of one, more, or none of the following flags:

```
memDebugModeCheckOnChange
memDebugModeCheckOnAll
memDebugModeScrambleOnChange
memDebugModeScrambleOnAll
memDebugModeFillFree
memDebugModeAllHeaps
memDebugModeRecordMinDynHeapFree
```

Returns Returns 0 if no error, or -1 if an error occurs.

MemStoreInfo Function

Purpose Return information on either the RAM store or the ROM store for a memory card.

Declared In `MemoryMgr.h`

Prototype

```
Err MemStoreInfo (UInt16 cardNo,
   UInt16 storeNumber, UInt16 *versionP,
   UInt16 *flagsP, Char *nameP, UInt32 *crDateP,
   UInt32 *bckUpDateP, UInt32 *heapListOffsetP,
   UInt32 *initCodeOffset1P,
   UInt32 *initCodeOffset2P, LocalID
   *databaseDirIDP)
```

Parameters

→ *cardNo*
Card number, either 0 or 1.

→ *storeNumber*
Store number; 0 for ROM, 1 for RAM.

↔ *versionP*
Pointer to version variable, or 0.

↔ *flagsP*
Pointer to flags variable, or 0.

↔ *nameP*
Pointer to character array (32 bytes), or 0.

↔ *crDateP*
Pointer to creation date variable, or 0.

↔ *bckUpDateP*
Pointer to backup date variable, or 0.

↔ *heapListOffsetP*
Pointer to `heapListOffset` variable, or 0.

↔ *initCodeOffset1P*
Pointer to `initCodeOffset1` variable, or 0.

↔ *initCodeOffset2P*
Pointer to `initCodeOffset2` variable, or 0.

↔ *databaseDirIDP*
Pointer to database directory chunk ID variable, or 0.

Returns Returns 0 if no error, or `memErrCardNotPresent`, `memErrRAMOnlyCard`, or `memErrInvalidStoreHeader` if an error occurs.

Comments Call this routine to retrieve any or all information on either the RAM store or the ROM store for a card. Pass 0 for variables that you don't wish returned.

43

Notification Manager

This chapter describes the Notification Manager API as declared in the header file `NotifyMgr.h`. It discusses the following topics:

- Notification Constants
- Notification Functions
- Application-Defined Functions

The chapter "Notifications" in this book lists the possible notifications and describes the data sent with each. Also see the section "Notifications" on page 30 in the "Application Startup and Stop" chapter of the *Palm OS Programmer's Companion*, vol. I for a description of how to use notifications.

Notification Constants

Miscellaneous Constants

This following miscellaneous constants are used in the Notification Manager. For other notification constants, see the "Notifications" chapter in this book.

Constant	Value	Description
`sysNotifyBroadcasterCode`	`'psys'`	The value passed as the creator ID of the broadcaster for notifications broadcast by the system.
`sysNotifyDefaultQueueSize`	30	The maximum number of nested broadcasts allowed.
`sysNotifyNoDatabaseID`	0xFFFFFFFF	The database local ID used by the system when it registers for notifications.

Constant	Value	Description
`sysNotifyNormalPriority`	0	Typical priority value used when registering for notifications.
`sysNotifyVersionNum`	1	Current Notification Manager version. This number is stored in the system feature `sysFtrNumNotifyMgrVersion`.

Notification Functions

SysNotifyBroadcast Function

Purpose Synchronously send a notification to all applications registered for it.

Declared In `NotifyMgr.h`

Prototype

```
Err SysNotifyBroadcast
    (SysNotifyParamType *notify)
```

Parameters

↔ *notify*
: Identifies the notification to be broadcast. See `SysNotifyParamType`.

Returns Returns one of the following error codes:

`errNone`
: No error.

`sysNotifyErrBroadcastBusy`
: The broadcast stack limit has already been reached.

`sysErrParamErr`
: The background thread is broadcasting the notification and the `notify` parameter is `NULL`.

`sysNotifyErrNoStackSpace`
: There is not enough space on the stack for the notification.

Comments When you call this function, the notification you specify is broadcast to all applications, shared libraries, and other code resources that have registered to receive that notification. The broadcast is

performed synchronously, meaning that the system broadcasts the notification immediately and waits for each notification client to perform its notification handler and return before the `SysNotifyBroadcast` call returns. This notification occurs in priority order.

The system allows nested notifications; that is, the recipient of a notification might broadcast a new notification, whose recipient might broadcast another new notification and so on. The constant `sysNotifyDefaultQueueSize` specifies how many levels of nested notification are allowed. If you reach this limit, the error `sysNotifyErrBroadcastBusy` is returned and your notification is not broadcast. To avoid reaching the limit, use `SysNotifyBroadcastDeferred()` instead of `SysNotifyBroadcast()` in your notification handlers. This ensures that the notification will not be broadcast until the top of the event loop.

WARNING! Do not call `SysNotifyBroadcast` from code that might be called from a background task (such as a trap patch) with the memory semaphore reserved. Use `SysNotifyBroadcastDeferred` instead.

Compatibility Implemented only if Notification Feature Set is present.

SysNotifyBroadcastDeferred Function

Purpose Enqueue a notification for later broadcast.

Declared In `NotifyMgr.h`

Prototype

```
Err SysNotifyBroadcastDeferred
    (SysNotifyParamType *notify, Int16 paramSize)
```

Parameters

↔ *notify*
The notification to enqueue. See `SysNotifyParamType`.

→ *paramSize*
Size of the data pointed to by the field `notify->notifyDetailsP`.

Returns Returns one of the following error codes:

`errNone`
No error.

`memErrNotEnoughSpace`
There is not enough memory to allocate a new notification entry in the queue.

`sysErrParamErr`
`paramSize` is a negative number.

`sysNotifyErrQueueFull`
The queue has reached its maximum number of entries.

Comments This function is similar to `SysNotifyBroadcast()` except that the broadcast does not take place until the top of the event loop (specifically, the next time `EvtGetEvent()` is called). The system copies the `notify` structure to a new memory chunk, which is disposed of upon completion of the broadcast. (The `paramSize` value is used when copying the `notifyDetailsP` portion of the `notify` structure.)

Compatibility Implemented only if Notification Feature Set is present.

SysNotifyBroadcastFromInterrupt Function

Purpose Allows interrupt handlers to enqueue a notification for later broadcast.

Declared In `NotifyMgr.h`

Prototype

```
Err SysNotifyBroadcastFromInterrupt
   (UInt32 notifyType, UInt32 broadcaster,
   void *notifyDetailsP)
```

Parameters

→ *`notifyType`*
The type of event that occurred. See the chapter Notifications for a complete list of the notifications that Palm OS® broadcasts.

→ *`broadcaster`*
The creator ID of the device or application that broadcast the notification.

→ *notifyDetailsP*
Pointer to data specific to this notification. See the Notifications chapter for the specific instances where this parameter is used.

Returns Returns one of the following error codes:

`errNone`
No error.

`sysNotifyErrQueueFull`
The queue has reached its maximum number of entries.

Comments Like `SysNotifyBroadcastDeferred()`, this function enqueues a notification to be broadcast at the top of the event loop (specifically, the next time `EvtGetEvent()` is called). It differs from `SysNotifyBroadcastDeferred` in that it is interrupt-safe and intended to be called from interrupt handlers.

This function is intended to be used by device drivers and other low-level software to generate a notification about a hardware change. For example, the Expansion Manager uses `SysNotifyBroadcastFromInterrupt` to broadcast `sysNotifyCardInsertedEvent` and `sysNotifyCardRemovedEvent` when a card is inserted into or removed from the expansion slot. `SysNotifyBroadcastFromInterrupt` is not intended to be used by general third party applications. Patching `SysNotifyBroadcastFromInterrupt` will cause the system to hang.

Compatibility Implemented only if 4.0 New Feature Set is present. If Palm OS Cobalt Feature Set is present this function is unimplemented.

WARNING! Using this function on Palm OS Garnet may cause a system crash.

SysNotifyRegister Function

Purpose Register to receive a notification.

Declared In `NotifyMgr.h`

Prototype

```
Err SysNotifyRegister (UInt16 cardNo,
   LocalID dbID, UInt32 notifyType,
   SysNotifyProcPtr callbackP, Int8 priority,
   void *userDataP)
```

Parameters

→ *cardNo*
Number of the storage card on which the application or code resource resides.

→ *dbID*
Local ID of the application or code resource.

→ *notifyType*
The notification that the application wants to receive. See the chapter Notifications.

→ *callbackP*
Set to `NULL` to receive the notification as an application launch code. If your code does not have a `PilotMain()` function (for example, if it is a shared library), pass a pointer to a function that should be called when the notification is broadcast. See `SysNotifyProcPtr()`.

→ *priority*
The priority with which the application should receive the event. Most applications and other code resources should always use `sysNotifyNormalPriority`. In rare circumstances, you may need to ensure that your code is notified toward the beginning or toward the end of the notification sequence. If so, be sure to leave some space so that your code won't collide with the system's handling of notifications or with another application's handling of notifications. In general, Palm™ recommends using a value whose least significant bits are 0 (such as 32, 64, 96, and so on). The smaller the priority, the earlier your code is notified.

→ *userDataP*
Caller-defined data to pass to the notification handler.

Returns Returns one of the following error codes:

`errNone`
: No error.

`sysErrParamErr`
: The database ID is `NULL`.

`sysNotifyErrDuplicateEntry`
: This application is already registered to receive this notification.

Comments

Call this function when your code should receive a notification that a specific event has occurred or is about to occur. See the Notifications chapter for a list of the possible notifications. Once you register for a notification, you remain registered to receive it until a system reset occurs or until you explicitly unregister using `SysNotifyUnregister()`.

If you're writing an application, you should pass `NULL` as the `callbackP` parameter. In this case, the system notifies your application by sending it the `sysAppLaunchCmdNotify` launch code. This launch code's parameter block points to a `SysNotifyParamType` structure containing details about the notification.

If your code is not in an application, for example, it is a shared library or a separately loaded code resource, then receiving a launch code is not possible. In this case, pass a pointer to a callback function in `callbackP`. This callback should follow the prototype shown in `SysNotifyProcPtr()`. Note that you should always supply a card number and database ID to `SysNotifyRegister`, even if you specify a callback function.

IMPORTANT: Because the `callbackP` pointer is used to directly call the function, the pointer must remain valid from the time `SysNotifyRegister` is called to the time the notification is broadcast. If the function is in a shared library, you must keep the library open. If the function is in a separately loaded code resource, the resource must remain locked while registered for the notification. When you close a library or unlock a resource, you must first unregister for any notifications. If you don't, the system will crash when the notification is broadcast.

Whether the notification handler is responding to `sysAppLaunchCmdNotify` or uses the callback function, the notification handler may perform any processing necessary. As with most launch codes, it's not possible to access global variables. If the handler needs access to any particular value to respond to the notification, pass a pointer to that value in the `userDataP` parameter. The system passes this pointer back to your application or callback function in the launch code's parameter block.

The notification handler may unregister for this notification or register for other notifications. It may also broadcast another notifications; however, it's recommended that you use `SysNotifyBroadcastDeferred()` to do this so as not to overflow the broadcast stack.

You may display a user interface in your notification handler; however, you should be careful when you do so. Many of the notifications are broadcast during `SysHandleEvent()`, which means your application event loop might not have progressed to the point where it is possible for you to display a user interface, or you might overflow the stack by displaying a user interface at this stage. See the "Notifications" chapter to learn which notifications are broadcast during `SysHandleEvent`.

Compatibility Implemented only if Notification Feature Set is present.

SysNotifyUnregister Function

Purpose Cancel notification of the given event.

Declared In `NotifyMgr.h`

Prototype

```
Err SysNotifyUnregister(UInt16 cardNo,
   LocalID dbID, UInt32 notifyType,
   Int8 priority)
```

Parameters

→ *cardNo*
Number of the storage card on which the application or code resource resides.

→ *dbID*
Local ID of the application or code resource.

→ *notifyType*
The notification for which to unregister. See Notifications.

→ *priority*
The priority value you passed as part of `SysNotifyRegister()`.

Returns Returns one of the following error codes:

`errNone`
No error.

`sysNotifyErrEntryNotFound`
The application wasn't registered to receive this notification.

Comments Use this function to remove your code from the list of those that receive notifications about a particular event. This function is particularly necessary if you are writing a shared library or a separately loaded code resource that receives notifications. When the resource is unloaded, it must unregister for all of its notifications, or the system will crash when the notification is broadcast.

Compatibility Implemented only if Notification Feature Set is present.

Application-Defined Functions

SysNotifyProcPtr Function

Purpose Handle a notification.

Declared In `NotifyMgr.h`

Prototype

```
Err (*SysNotifyProcPtr)
    (SysNotifyParamType *notifyParamsP)
```

Parameters → *notifyParamsP*
Pointer to a structure that contains the notification event that occurred and any other information about it. See `SysNotifyParamType`.

Returns Always return 0.

Comments You pass this function as a parameter to `SysNotifyRegister()` when registering code that does not have a `PilotMain()` for a

notification. See the description of `SysNotifyRegister()` for advice on writing a notification handler.

IMPORTANT: Because the `callbackP` pointer is used to directly call the function, the pointer must remain valid from the time `SysNotifyRegister` is called to the time the notification is broadcast. If the function is in a shared library, you must keep the library open. If the function is in a separately loaded code resource, the resource must remain locked while registered for the notification. When you close a library or unlock a resource, you must first unregister for any notifications. If you don't, the system will crash when the notification is broadcast.

44

Overlay Manager

This chapter describes the overlay manager API as declared in the header file `OverlayMgr.h`. It discusses the following topics:

- Overlay Manager Data Structures
- Overlay Manager Constants
- Overlay Manager Functions

For more information on the overlay manager, see "Using Overlays to Localize Resources" on page 387 of the *Palm OS Programmer's Companion*, vol. I.

Overlay Manager Data Structures

OmLocaleType Struct

Purpose The `OmLocaleType` struct specifies a locale.

Prototype

```
typedef struct {
  UInt16 language;
  UInt16 country;
} OmLocaleType;
```

Fields

`language`
: The language spoken in the locale. This value is one of the `LanguageType` constants.

`country`
: The country or region where the language is spoken. This value is one of the `CountryType` constants.

Compatibility If Palm OS® 4.0 New Feature Set is present, the `LmLocaleType` replaces `OmLocaleType`. For backward compatibility, `OmLocaleType` is mapped to `LmLocaleType`.

Overlay Manager Constants

Constant	Value	Description
`omOverlayRscType`	`'ovly'`	Symbolic name of an overlay resource that is contained in both the base database and the overlay database.
`omOverlayRscID`	1000	Resource ID of the overlay resource in both the base database and the overlay database.
`omFtrCreator`	`'ovly'`	Creator value used for the `omFtrShowErrorsFlag` feature.
`omFtrDefaultLocale`	1	Feature that specifies the default locale stored in the ROM. The default locale is used in cases where the system is attempting to open a "stripped" database (one that requires an overlay) and an overlay matching the current locale cannot be found. In this case, the system then looks for an overlay matching the default locale. Use `FtrGet()` and `FtrSet()` to retrieve and set this value.
`omFtrShowErrorsFlag`	0	Feature that controls the number of error messages displayed by the overlay manager. If this feature is set to `true`, the overlay manager may display several more error messages when validating an overlay against its base database. This feature only takes effect when the error checking level is set to full (common on debug ROMs, not on release ROMs). Use `FtrGet()` and `FtrSet()` to retrieve and set this value.

Overlay Manager Functions

OmGetCurrentLocale Function

Purpose Return the current locale.

Declared In `OverlayMgr.h`

Prototype

```
void OmGetCurrentLocale
    (LmLocaleType *currentLocale)
```

Parameters

← *currentLocale*
Points to an `LmLocaleType` struct that identifies the current locale.

Returns Returns nothing.

Comments This function returns the current locale. The current locale controls which overlays are used for resource databases. For example, suppose you have one application and two associated overlays installed, one for US English and one for British English. In this case, if the country specified in the locale returned by this function is `cUnitedKingdom`, the British English overlay is used for the application. If the country returned is `cUnitedStates`, the US English overlay is used.

Compatibility Implemented only if 3.5 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `OmGlueGetCurrentLocale`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `OmGetSystemLocale()`

OmGetIndexedLocale Function

Purpose Return a system locale by index.

Declared In `OverlayMgr.h`

Prototype

```
Err OmGetIndexedLocale (UInt16 localeIndex,
   LmLocaleType *theLocale)
```

Parameters

→ *`localeIndex`*
Zero-based index of the locale to return.

← *`theLocale`*
Points to an `LmLocaleType` struct that identifies the locale at that index.

Returns Returns `errNone` upon success, or `omErrInvalidLocaleIndex` if the index is out of bounds.

Comments `OmGetIndexedLocale` is used in a loop to discover how many system overlays are installed for system resources.

If the 4.0 New Feature Set is present, use `OmGetNextSystemLocale()` instead of this function. `OmGetIndexedLocale` can be slow on ROMs that contain many valid system locales.

Compatibility Implemented only if 3.5 New Feature Set is present. If Palm OS Cobalt Feature Set is present, however, this function is unimplemented.

In Palm OS 3.5, this function would not return a system overlay that was located in RAM. The Palm OS 4.0 version of this function does return system overlays located in RAM.

See Also `OmGetSystemLocale()`, `OmGetNextSystemLocale()`

OmGetNextSystemLocale Function

Purpose Return a system locale.

Declared In `OverlayMgr.h`

Prototype

```
Err OmGetNextSystemLocale (Boolean iNewSearch,
    OmSearchStateType *ioStateInfoP,
    LmLocaleType *oLocaleP)
```

Parameters

→ *iNewSearch*
: `true` if this function call is starting a new search, or `false` if this function call is a continuation of a search.

↔ *ioStateInfoP*
: If `iNewSearch` is `false`, this must point to the same data used for the previous invocation.

← *oLocaleP*
: The found locale.

Returns Returns `errNone` if no error or `omErrNoNextSystemLocale` if no matches were found.

Comments You can call this function successively to discover how many system overlays are installed for system resources. Each system overlay found determines a separate valid system locale. Any locale returned by this function can be sent to `OmSetSystemLocale()` to change the system locale.

To start the search, pass `true` for `iNewSearch`. Allocate an `OmSearchStateType` structure and pass its address as `ioStateInfoP`. `OmGetNextSystemLocale` stores private information in `ioStateInfoP` and uses it if the search is continued.

To continue a search where the previous one left off, pass `false` for `iNewSearch` and pass the same `ioStateInfoP` that you used during the previous call to this function.

When called successively, this function eventually returns all system overlays that are in ROM or RAM.

Compatibility Implemented only if 4.0 New Feature Set is present. If Palm OS Cobalt Feature Set is present, however, this function is unimplemented.

OmGetRoutineAddress Function

Purpose Return the address of an overlay manager function.

Declared In `OverlayMgr.h`

Prototype

```
void *OmGetRoutineAddress (OmSelector inSelector)
```

Parameters → *inSelector*
One of the routine selectors defined in `OverlayMgr.h`.

Returns Returns the address of the corresponding function. Returns `NULL` if an invalid routine selector is passed.

Comments You typically use this function to determine whether an overlay manager function exists on the device. As future releases of Palm OS add new functions, older devices with earlier versions of the overlay manager will not implement these newer functions. If `OmGetRoutineAddress` returns `NULL`, the function is unavailable.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `IntlGetRoutineAddress()`, `SysGetTrapAddress()`

OmGetSystemLocale Function

Purpose Return the system locale.

Declared In `OverlayMgr.h`

Prototype

```
void OmGetSystemLocale
    (LmLocaleType *systemLocale)
```

Parameters ← *systemLocale*
Points to an `LmLocaleType` struct that identifies the system locale.

Returns Returns nothing.

Comments You typically don't use this function. Instead, use `OmGetCurrentLocale()`, which returns the locale that determines which overlays are used.

The system locale is saved in the storage heap header and persists across soft resets. When the device is reset, the system locale is used to set the current locale.

Compatibility Implemented only if 3.5 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `OmGlueGetSystemLocale`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also OmGetCurrentLocale(), LmGlueGetSystemLocale()

OmLocaleToOverlayDBName Function

Purpose Return the overlay database's name given the base database name and the locale.

Declared In `OverlayMgr.h`

Prototype

```
Err OmLocaleToOverlayDBName
    (const Char *baseDBName,
    const LmLocaleType *targetLocale,
    Char *overlayDBName)
```

Parameters

→ *`baseDBName`*
The name of the base resource database associated with the overlay.

→ *`targetLocale`*
The locale to which this overlay applies. See `LmLocaleType`. Pass `NULL` to use the current locale.

← *`overlayDBName`*
The overlay database name given the base database name and the target locale. This buffer must be at least `dmDBNameLength` bytes.

Returns Returns `errNone` upon success, or `omErrUnknownLocale` if the `targetLocale` parameter is invalid.

Comments The appropriate overlay database name is currently:

`baseDBName_llCC`

where:

`baseDBName`
The name of the base database as you passed it in.

`ll`
A two-character code identifying the language.

CC
A two-character code identifying the country.

The base database name is truncated if necessary to allow for this suffix.

For example, the base database "MemoPad" might have an overlay for US English named "MemoPad_enUS".

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `OmOverlayDBNameToLocale()`

OmOverlayDBNameToLocale Function

Purpose Return an overlay database's locale given its name.

Declared In `OverlayMgr.h`

Prototype

```
Err OmOverlayDBNameToLocale
    (const Char *overlayDBName,
    LmLocaleType *overlayLocale)
```

Parameters

→ *overlayDBName*
The name of the overlay database.

← *overlayLocale*
Points to an `LmLocaleType` structure identifying the overlay's locale.

Returns Returns `errNone` upon success, `omErrBadOverlayDBName` if the string `overlayDBName` is not long enough to have a locale suffix, or `omErrUnknownLocale` if the locale cannot be determined.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `OmLocaleToOverlayDBName()`

OmSetSystemLocale Function

Purpose Set the system locale and reset the device.

Declared In `OverlayMgr.h`

Prototype

```
Err OmSetSystemLocale
    (const LmLocaleType *systemLocale)
```

Parameters

→ *systemLocale*
An `LmLocaleType` structure specifying the locale to switch the system to.

Returns Returns `errNone` upon success, or one of the following if an error occurs:

`omErrUnknownLocale`
There is no system overlay for `systemLocale`.

`omErrInvalidLocale`
The system overlay for `systemLocale` has been found but is invalid.

`dmErrInvalidParam`
An error occurred while opening the overlay.

`dmErrMemError`
A memory error occurred while opening the overlay.

`dmErrDatabaseOpen`
The system overlay was already open.

Comments This function changes the system locale to the specified locale if it exists. It first determines that the system overlay exists for the requested locale and that it matches the base system database. If so, it updates the system locale information saved in the storage heap header and resets the device. After the device is reset, the current locale is set to the system locale.

A Palm Powered™ device has a default locale hard-coded into the ROM. This locale is used to set the system locale after a hard reset or any time that the storage heap header is invalid. The storage heap header is typically only invalid when the device is turned on for the first time.

Compatibility Implemented only if 3.5 New Feature Set is present. If Palm OS Cobalt Feature Set is present, however, this function is unimplemented.

In Palm OS 3.5, this function would not switch to a system overlay that was located in the RAM. The Palm OS 4.0 version of this function does return system overlays located in the RAM.

See Also `OmGetSystemLocale()`

45

Password

This chapter provides reference material for the password API. The password API is declared in the header file `Password.h`.

Password Functions

PwdExists Function

Purpose Return `true` if the system password is set.

Declared In `Password.h`

Prototype `Boolean PwdExists()`

Parameters None

Returns Returns `true` if the system password is set.

PwdRemove Function

Purpose Remove the encrypted password string and recover data hidden in databases.

Declared In `Password.h`

Prototype `void PwdRemove (void)`

Parameters None

Returns Returns nothing.

PwdSet Function

Purpose Use a passed string as the new password. The password is stored in an encrypted form.

Declared In `Password.h`

Prototype

```
void PwdSet (Char *oldPassword, Char
    *newPassword)
```

Parameters

`oldPassword`
: The old password must be successfully verified or the new password isn't accepted

`newPassword`
: `Char*` to a string to use as the password. `NULL` means no password.

Returns Returns nothing.

PwdVerify Function

Purpose Verify that the string passed matches the system password.

Declared In `Password.h`

Prototype

```
Boolean PwdVerify (Char *string)
```

Parameters

`string`
: String to compare to the system password. `NULL` means no current password.

: The allocated length of string must be at least `pwdLength` characters long.

Returns Returns `true` if the string matches the system password.

46

Pen Input Manager Reference

This chapter provides reference material for the Pen Input Manager API as declared in the header file `PenInputMgr.h`. It discusses the following topics:

- Pen Input Manager Constants
- Pen Input Manager Functions
- Control Bar Functions
- Display Orientation Functions

Pen Input Manager Constants

Input Area States

Table 46.1 lists constants that define the states that the input area can have. An application can obtain the input area's current state with `PINGetInputAreaState` and set it with `PINSetInputAreaState`.

Table 46.1 Input area states

Constant	Value	Description
`pinInputAreaOpen`	0	The dynamic input area is being displayed.
`pinInputAreaClosed`	1	The dynamic input area is not being displayed.

Table 46.1 Input area states *(continued)*

Constant	Value	Description
		The dynamic input area is in this state after the user taps the input trigger to close it. An application also might request that the dynamic input area be closed by calling `PINSetInputAreaState` with this state.
`pinInputAreaNone`	2	The input area is not dynamic, or there is no input area. Do not pass this value to `PINSetInputAreaState`.
`pinInputAreaUser`	5	Pass this value to `PINSetInputAreaState` to tell the Pen Input Manager to activate the last user-selected input area state.

Input Trigger States

Table 46.2 lists constants that specify the state of the input area icon in the status bar. An application can obtain this state with `PINGetInputTriggerState` and set it with `PINSetInputTriggerState`.

Table 46.2 Input trigger states

Constant	Value	Description
`pinInputTriggerEnabled`	0	The input trigger is enabled, meaning that the user is allowed to open and close the dynamic input area.
`pinInputTriggerDisabled`	1	The input trigger is disabled, meaning that the user is not allowed to close the dynamic input area.
`pinInputTriggerNone`	2	There is no dynamic input area.

Orientation States

Table 46.3 lists constants that specify the display orientation. An application can obtain this state with `SysGetOrientation` and set it with `SysSetOrientation`.

Table 46.3 Orientation state constants

Constant	Value	Description
`sysOrientationUser`	0	Pass this value to `SysSetOrientation` to tell the system to activate the last user-selected orientation.
`sysOrientationPortrait`	1	The display is in portrait orientation.
`sysOrientationLandscape`	2	The display is in landscape orientation.
`sysOrientationReversePortrait`	3	The display is in reverse portrait orientation (upside-down from the normal portrait orientation).
`sysOrientationReverseLandscape`	4	The display is in reverse landscape orientation (upside-down from the normal landscape orientation).

Orientation Trigger States

Table 46.4 lists constants that specify the state of the orientation icon in the status bar (the icon that allows the user to change the display orientation). An application can obtain this state with `SysGetOrientationTriggerState` and set it with `SysSetOrientationTriggerState`.

Table 46.4 Orientation trigger state constants

Constant	Value	Description
`sysOrientationTriggerDisabled`	0	The orientation trigger is disabled, meaning that the user is not allowed to change the display orientation.
`sysOrientationTriggerEnabled`	1	The orientation trigger is enabled, meaning that the user is allowed to change the display orientation.

Pen Input Manager Functions

PINGetInputAreaState Function

Purpose Returns the current state of the dynamic input area.

Declared In `PenInputMgr.h`

Prototype
```
UInt16 PINGetInputAreaState (void)
```

Parameters `None.`

Returns One of the constants defined in the section "Input Area States" on page 881.

Compatibility Implemented only if the Pen Input Manager Feature Set is present.

See Also `PINSetInputAreaState`, `PINGetInputTriggerState`

PINGetInputTriggerState Function

Purpose Returns the status of the input area icon in the status bar.

Prototype
```
UInt16 PINGetInputTriggerState (void)
```

Parameters None.

Returns One of the constants defined in the section "Input Trigger States" on page 882.

Compatibility Implemented only if the Pen Input Manager Feature Set is present.

See Also `PINGetInputAreaState`

PINSetInputAreaState Function

Purpose Sets the state of the input area.

Declared In `PenInputMgr.h`

Prototype

```
Err PINSetInputAreaState (UInt16 state)
```

Parameters

→ *state*
The state to which the input area should be set. See "Input Area States" on page 881 for a list of possible values.

Returns Returns one of the following error codes:

`errNone`
Success.

`pinErrNoSoftInputArea`
There is no dynamic input area on this device.

`pinErrInvalidParam`
You have entered an invalid `state` parameter.

Comments After opening or closing the input area, this function broadcasts the notification `sysNotifyDisplayResizedEvent` (and in Pen Input Manager version 1.1, posts the event `winDisplayChangedEvent` to the event queue). Applications register for this notification or respond to the event if they wish to resize themselves.

Compatibility Implemented only if the Pen Input Manager Feature Set is present.

See Also `PINGetInputAreaState`, `PINSetInputTriggerState`

PINSetInputTriggerState Function

Purpose Sets the state of the input area icon in the status bar.

Declared In `PenInputMgr.h`

Prototype

```
Err PINSetInputTriggerState (UInt16 state)
```

Parameters

→ *`state`*
The state to which the input trigger should be set. See "Input Trigger States" on page 882 for a list of possible values.

Returns Returns one of the following error codes:

`errNone`
Success.

`pinErrNoSoftInputArea`
There is no dynamic input area on this device.

`pinErrInvalidParam`
You have specified an invalid `state` parameter.

Comments Applications or Palm OS call this function to enable the input area icon in the status bar. Normally, this trigger is enabled and should remain enabled, allowing the user the choice of displaying the input area or not. Legacy applications might disable the trigger on some devices.

Compatibility Implemented only if the Pen Input Manager Feature Set is present.

See Also `PINGetInputTriggerState`, `PINSetInputAreaState`

Control Bar Functions

This section lists other functions that are implemented as part of the Pen Input Manager in Palm OS Garnet.

StatBarGetIcon Function

Purpose Obtain the identifier for the status bar icon currently displayed in a specified slot.

Declared In `PenInputMgr.h`

Prototype

```
Boolean StatBarGetIcon (UInt16 slotID,
    UInt16 *iconIDPtr)
```

Parameters

→ *slotID*
Unique identifier of the status bar slot containing the desired icon.

← *iconIDPtr*
The ID of the bitmap image displayed in the specified slot. This identifier represents one of the bitmap images in the device resources database.

Returns `true` if the slot was found, `false` if it is not one of the slots currently being kept track of.

StatBarSetIcon Function

Purpose Changes the icon state of a specified status bar icon.

Declared In `PenInputMgr.h`

Prototype

```
Err StatBarSetIcon (UInt16 slotID, UInt16 iconID)
```

Parameters

→ *slotID*
Unique identifier of the status bar slot containing the icon to be changed.

→ *iconID*
Identifier of one of the bitmap images in the device resources database. The specified image will be displayed in the slot indicated by *slotID*.

Returns Returns `errNone` if the status bar icon was updated successfully, or an error otherwise. The most common error returned from this function arises when you attempt to add an icon to a status bar that already contains the maximum number of icons.

StatGetAttribute Function

Purpose Returns the control bar state.

Declared In `PenInputMgr.h`

Prototype

```
Err StatGetAttribute (UInt16 selector,
   UInt32 *dataP)
```

Parameters

→ *`selector`*
Attribute selector, as described in the Comments section below.

→ *`dataP`*
Pointer to the returned data, as described in the Comments section below.

Returns Returns one of the following error codes:

`errNone`
Success.

`sysErrParamErr`
You have specified an invalid `selector` parameter.

Comments The following values are supported for the selector parameter:

`statAttrBarVisible`
Checks if the control bar is visible. The return data is set to 0 if the control bar is hidden, or is set to 1 if the control bar is visible or the input area is open.

`statAttrDimension`
Gets the control bar bounds. The return data is two `UInt16` values, where the first is the width of the control bar and the second is the height. The dimensions use the active coordinate system.

Compatibility Implemented only if the Pen Input Manager Feature Set is present and returns version 1.1.

See Also `StatHide`

StatHide Function

Purpose Hides the control bar.

Declared In `PenInputMgr.h`

Prototype

```
Err StatHide (void)
```

Returns Returns one of the following error codes:

`errNone`
: Success.

`sysErrNotAllowed`
: The device does not support a dynamic input area.

`sysErrInputWindowOpen`
: The input area is open (so the control bar is not currently visible).

Comments The input area must be closed before you call this function.

This function can be called by applications that want to draw to the entire display area and thus need to hide the control bar. However, hiding the control bar is discouraged since it prevents users from exiting to the Launcher or opening the input area via buttons that appear on the control bar. If the control bar is hidden, you must provide a mechanism for the user to exit the application.

Compatibility Implemented only if the Pen Input Manager Feature Set is present and returns version 1.1.

See Also `StatGetAttribute`, `StatShow`

StatShow Function

Purpose Shows the control bar.

Declared In `PenInputMgr.h`

Prototype

```
Err StatShow (void)
```

Returns Returns one of the following error codes:

`errNone`
: Success.

`sysErrNotAllowed`
: The device does not support a dynamic input area.

Comments If the input area is open when this function is called, it has no effect and `errNone` is returned.

Compatibility Implemented only if the Pen Input Manager Feature Set is present and returns version 1.1.

See Also `StatGetAttribute`, `StatHide`

Display Orientation Functions

SysGetOrientation Function

Purpose Returns the display orientation.

Declared In `SystemMgr.h`

Prototype `UInt16 SysGetOrientation (void)`

Parameters None.

Returns Returns one of the constants listed in "Orientation States" on page 883.

Comments Not all devices support changing the display orientation. For devices that don't support changing the display orientation, this function always returns `sysOrientationPortrait`.

Compatibility Implemented only if Pen Input Manager Feature Set is present and returns version 1.1. Some licensees may have implemented this function in an earlier OS version. To check if this function is implemented in an earlier OS version, use this test:

```
if (SysGlueTrapExists(pinSysGetOrientation)) {
// SysGetOrientation exists
}
```

See Also `SysSetOrientation`

SysGetOrientationTriggerState Function

Purpose Returns the display orientation trigger state.

Declared In `SystemMgr.h`

Prototype

```
UInt16 SysGetOrientationTriggerState (void)
```

Returns Returns one of the constants listed in "Orientation Trigger States" on page 883.

Comments Not all devices support changing the display orientation. For devices that don't support changing the display orientation, this function always returns `sysOrientationTriggerDisabled`.

Compatibility Implemented only if the Pen Input Manager Feature Set is present and returns version 1.1. Some licensees may have implemented this function in an earlier OS version. To check if this function is implemented in an earlier OS version, use this test:

```
if (SysGlueTrapExists (pinSysGetOrientationTriggerState)) {
// SysGetOrientationTriggerState exists
}
```

See Also `SysSetOrientationTriggerState`

SysSetOrientation Function

Purpose Sets the display orientation.

Declared In `SystemMgr.h`

Prototype

```
Err SysSetOrientation (UInt16 orientation)
```

Parameters

→ *orientation*
The orientation to which the display should be set. See "Orientation States" on page 883 for a list of possible values.

Returns Returns one of the following error codes:

`errNone`
Success.

`sysErrNotAllowed`
Setting the display orientation is not supported on the device.

Comments Not all devices support changing the display orientation.

Compatibility Implemented only if the Pen Input Manager Feature Set is present and returns version 1.1. Some licensees may have implemented this function in an earlier OS version. To check if this function is implemented in an earlier OS version, use this test:

```
if (SysGlueTrapExists(pinSysSetOrientation)) {
// SysSetOrientation exists
}
```

See Also `SysGetOrientation`

SysSetOrientationTriggerState Function

Purpose Sets the display orientation trigger state.

Declared In `SystemMgr.h`

Prototype

```
Err SysSetOrientationTriggerState
    (UInt16 triggerState)
```

Parameters

→ *`triggerState`*
One of the constants listed in "Orientation Trigger States" on page 883.

Returns Returns one of the following error codes:

`errNone`
Success.

`sysErrNotAllowed`
Setting the display orientation is not supported on the device.

Comments Not all devices support changing the display orientation.

Compatibility Implemented only if the Pen Input Manager Feature Set is present and returns version 1.1. Some licensees may have implemented this function in an earlier OS version. To check if this function is implemented in an earlier OS version, use this test:

```
if (SysGlueTrapExists(pinSysSetOrientationTriggerState)) {
// SysSetOrientationTriggerState exists
}
```

See Also `SysGetOrientationTriggerState`

47

Pen Manager

This chapter provides reference material for the pen manager. The pen manager API is declared in the header file `PenMgr.h`.

For more information on the pen manager, see "The Pen Manager" on page 63 of the *Palm OS Programmer's Companion*, vol. I.

Pen Manager Functions

PenCalibrate Function

Purpose Set the calibration of the pen.

Declared In `PenMgr.h`

Prototype

```
Err PenCalibrate (PointType *digTopLeftP,
   PointType *digBotRightP,
   PointType *scrTopLeftP,
   PointType *scrBotRightP)
```

Parameters

`digTopLeftP`
: Digitizer output from top-left coordinate.

`digBotRightP`
: Digitizer output from bottom-right coordinate.

`scrTopLeftP`
: Screen coordinate near top-left corner.

`scrBotRightP`
: Screen coordinate near bottom-right corner.

Returns Returns 0 if no error.

Comments Called by Preferences application when calibrating pen.

Compatibility If Palm OS Cobalt Feature Set is present this function does nothing.

See Also PenResetCalibration()

PenResetCalibration Function

Purpose Reset the calibration in preparation for calibrating the pen again.

Declared In `PenMgr.h`

Prototype

```
Err PenResetCalibration (void)
```

Parameters None.

Returns Always returns 0.

Comments Called by Preferences application before capturing points when calibrating the pen.

WARNING! The digitizer is off after calling this routine and must be calibrated again!

Compatibility If Palm OS Cobalt Feature Set is present this function does nothing.

See Also PenCalibrate()

48

Preferences

This chapter describes the preferences API as declared in the header file `Preferences.h`. It discusses the following topics:

- Preferences Data Types
- Preferences Constants
- Preferences Functions

For more information on preferences, see the section "Preferences" on page 342 of the *Palm OS Programmer's Companion*, vol. I.

Preferences Data Types

MeasurementSystemType Enum

Purpose The `MeasurementSystemType` enum specifies the preference for units of measure.

Prototype

```
typedef enum {
  unitsEnglish = 0,
  unitsMetric
} MeasurementSystemType;
```

Constants

`unitsEnglish`
: The English measurement system (feet, inches, and so on).

`unitsMetric`
: The Metric system (meters, centimeters, and so on).

SecurityAutoLockType Enum

Purpose The `SecurityAutoLockType` enum specifies the values for the auto-locking preference. The auto-locking preference specifies when the device will shut down and lock itself.

Prototype

```
typedef enum {
  never = 0,
  uponPowerOff,
  atPresetTime,
  afterPresetDelay
} SecurityAutoLockType;
```

Constants

`never`
: The auto-lock feature is disabled.

`uponPowerOff`
: The device locks itself each time it is powered off.

`atPresetTime`
: The device locks itself at a certain time every day.

`afterPresetDelay`
: The device locks itself after a certain amount of idle time.

SoundLevelTypeV20 Enum

Purpose

The `SoundLevelTypeV20` enum specifies whether certain sounds are enabled or disabled. This enum is used as the values for several of the system sound preferences.

Prototype

```
typedef enum {
  slOn = 0,
  slOff = 1
} SoundLevelTypeV20;
```

Constants

`slOn`
: Enabled

`slOff`
: Disabled

Compatibility

This enumerated type is obsolete in Palm OS® versions 3.0 and higher. The preferences that use this enum are replaced by preferences that indicate sound volume as well as whether the sound is on or off.

SystemPreferencesChoice Enum

The `SystemPreferencesChoice` enum defines values that you can pass to `PrefGetPreference()` and `PrefSetPreference()` to retrieve or set a system preference value. `SystemPreferencesChoice` defines one constant for each field in the `SystemPreferencesType` structure, which should be considered a private structure.

Table 48.1 lists and describes the `SystemPreferencesChoice` constants. For each constant, it shows what type of data is returned by `PrefGetPreference` for that constant and which version of the system preferences structure contains this preference.

The system preferences structure keeps track of its own version information because it has been updated several times. Each Palm OS release that updates the system preferences structure increments the structure's version number. Table 48.1 on page 900 specifies which version of the system preferences structure introduced that preference. For each preferences version, there is a constant representing that version. See the section "Preferences Constants" on page 910 for information on which preferences version corresponds with which Palm OS release. You should check the preference's version number before attempting to obtain the value of any preference introduced after version 2. For example:

```
LmLocaleType currentLocale;
if (PrefGetPreference(prefVersion) >=
    preferenceDataVer9) {
  currentLocale = (LmLocaleType)
    PrefGetPreference(prefLocale);
}
```

Most of the system preferences can be set in the Preferences and Security applications. Table 48.2 on page 908 specifies which system preference is set by each user interface field in these two applications.

Table 48.1 SystemPreferencesChoice Constants

Constant	Type	Version	Description
`prefVersion`	`UInt16`	2	The preferences version number.
`prefCountry`	`CountryType`	2	The country for which the device was built.
`prefDateFormat`	`DateFormatType`	2	The short format used to display dates. For example: 95/12/31
`prefLongDateFormat`	`DateFormatType`	2	The long format used to display dates. For example: 31 Dec 1995
`prefWeekStartDay`	`Int8`	2	The first day of the week (Sunday or Monday). Days of the week are numbered from 0 to 6 starting with Sunday = 0.
`prefTimeFormat`	`TimeFormatType`	2	The format used to display time values.
`prefNumberFormat`	`NumberFormatType`	2	The format used for numbers, with regards to the thousands separator and the decimal point.
`prefAutoOffDuration`	`UInt8`	2	Minutes of user idle time before the device powers off. The default value for this preference is specified by the `defaultAutoOffDuration` constant.

Table 48.1 SystemPreferencesChoice Constants *(continued)*

Constant	Type	Version	Description
			`prefAutoOffDuration` is replaced by `prefAutoOffDurationSecs` in version 8 of the preferences structure.
`prefSysSoundLevelV20`	`SoundLevelTypeV20`	2	Specifies whether system sounds are enabled or disabled.
`prefGameSoundLevelV20`	`SoundLevelTypeV20`	2	Specifies whether game sound effects are on or off.
`prefAlarmSoundLevelV20`	`SoundLevelTypeV20`	2	Specifies whether sound alarms are on or off.
`prefHidePrivateRecordsV33`	`Boolean`	2	If `true`, applications should not display database records that have the secret attribute bit set.
`prefDeviceLocked`	`Boolean`	2	If `true`, the device is locked. When the device is locked, it remains so until the user enters the password.
`prefLocalSyncRequiresPassword`	`Boolean`	2	If `true`, the user must enter a password before a HotSync® operation can be performed.
`prefRemoteSyncRequiresPassword`	`Boolean`	2	If `true`, the user must enter a password on the desktop computer before a HotSync operation can be performed.
`prefSysBatteryKind`	`SysBatteryKind`	2	The type of batteries installed. Use `SysBatteryInfo()` to retrieve the battery type instead of this preference.

Table 48.1 SystemPreferencesChoice Constants *(continued)*

Constant	Type	Version	Description
`prefMinutesWestOfGMT`	`UInt32`	2	The time zone given as minutes *east* of Greenwich Mean Time (GMT). For preferences version 9 and higher, use `prefTimeZone` instead.
`prefDaylightSavings`	`DaylightSavingsTypes`	2	The type of daylight savings correction. For preferences version 9 and higher, use `prefDaylightSavingAdjustment` instead.
`prefRonamaticChar`	`UInt16`	2	The virtual character generated when the user enters the ronamatic stroke. The ronamatic stroke is made by dragging the pen from the input area to the top of the screen.
`prefHard1CharAppCreator`	`UInt32`	2	The creator ID of the application to be launched by the left-most hard key (the Date Book button by default).
`prefHard2CharAppCreator`	`UInt32`	2	The creator ID of the application to be launched by the second hard key from the left (the Address button by default).
`prefHard3CharAppCreator`	`UInt32`	2	The creator ID of the application to be launched by the second hard key from the right (the To Do List button by default).

Table 48.1 SystemPreferencesChoice Constants *(continued)*

Constant	Type	Version	Description
`prefHard4CharAppCreator`	`UInt32`	2	The creator ID of the application to be launched by the right-most hard key (the Memo Pad button by default).
`prefCalcCharAppCreator`	`UInt32`	2	The creator ID of the application to be launched by the Calculator silkscreen button.
`prefHardCradleCharAppCreator`	`UInt32`	2	The creator ID of the application to be launched by the hard key on the HotSync cradle.
`prefLauncherAppCreator`	`UInt32`	2	The creator ID of the application to be launched by the Applications silkscreen button.
`prefHardCradle2CharAppCreator`	`UInt32`	2	The creator ID of the application to be launched by the HotSync button on the modem.
`prefAnimationLevel`	`AnimationLevelType`	2	Reserved for future use.
`prefSysSoundVolume`	`UInt16`	3	The sound level for system sounds, such as taps and beeps. This is a value from 0 to `sndMaxAmp`.
`prefGameSoundVolume`	`UInt16`	3	The sound level for game sounds. This is a value from 0 to `sndMaxAmp`.

Table 48.1 SystemPreferencesChoice Constants *(continued)*

Constant	Type	Version	Description
`prefAlarmSoundVolume`	`UInt16`	3	The sound level for alarms. This is a value from 0 to `sndMaxAmp`.
`prefBeamReceive`	`Boolean`	3	If `true`, the device can receive beams from other devices. If `false`, the device cannot receive beams but can still send them. This preference is not currently used. Instead, use the `ExgControl()` function with one of the constants described in "IR Control Constants."
`prefCalibrateDigitizerAtReset`	`Boolean`	3	If `true`, the user must recalibrate the digitizer after a soft reset. The default is `false`.
`prefSystemKeyboardID`	`UInt16`	4	The resource ID of the keyboard panel.
`prefDefSerialPlugIn`	`UInt32`	4	The creator ID of the default serial plug-in database.
`prefStayOnWhenPluggedIn`	`Boolean`	5	If `true`, the device stays powered on when it is in the cradle.
`prefStayLitWhenPluggedIn`	`Boolean`	5	If `true` and `prefStayOnWhenPluggedIn` is `true`, the device stays lit when it is in its cradle.
`prefAntennaCharAppCreator`	`UInt32`	6	The creator ID of the application to launch when the antenna is raised (used only for devices with built-in antennas).

Table 48.1 SystemPreferencesChoice Constants *(continued)*

Constant	Type	Version	Description
`prefMeasurementSystem`	`MeasurementSystemType`	7	The system of measurement to use.
`prefShowPrivateRecords`	`privateRecordViewEnum`	8	Specifies whether the private records should be displayed, masked, or completely hidden.
`prefAutoOffDurationSecs`	`UInt16`	8	Seconds of user idle time before the device powers off. The default value for this preference is specified by the `defaultAutoOffDurationSecs` constant.
`prefTimeZone`	`Int16`	9	The time zone given as minutes east of Greenwich Mean Time (GMT). **IMPORTANT:** Changing the value of this preference does not update the current time.
`prefDaylightSavingAdjustment`	`Int16`	9	The number of minutes to add to the current time for daylight savings time. **IMPORTANT:** Changing the value of this preference does not update the current time.
`prefTimeZoneCountry`	`CountryType`	9	The country selected to specify what the time zone is.

Table 48.1 SystemPreferencesChoice Constants *(continued)*

Constant	Type	Version	Description
`prefAutoLockType`	`SecurityAutoLockType`	9	Specifies when the auto-locking feature should take effect. Possibilities are upon power off, at a preset time, or after a certain number of seconds.
`prefAutoLockTime`	`UInt32`	9	The time value for the auto-locking feature if the system should lock itself after a delay or at a preset time. Depending on the value of `prefAutoLockType`, this value is either an absolute date and time given as the number of seconds since January 1, 1904 or a timeout value given as a number of seconds from the current time.
`prefAutoLockTimeFlag`	`Boolean`	9	If `true`, `prefAutoLockTime` is given in minutes. If `false`, the time is given in hours.
`prefLanguage`	`LanguageType`	9	The language that the device should use.
`prefAttentionFlags`	`AttnFlagsType`	9	The user's preferences for receiving attention signals. The returned value is a bit mask that should be tested (using the `&` operator) with one of the following values:

Table 48.1 SystemPreferencesChoice Constants *(continued)*

Constant	Type	Version	Description

			`kAttnFlagsUserWantsLED`
			`kAttnFlagsUserWantsSound`
			`kAttnFlagsUserWantsVibrate`
			`kAttnFlagsUserWantsCustomEffect`
			Note that you can override the values in `prefAttentionFlags` when you make Attention Manager function calls. See the section "Getting the User's Attention" on page 305 of the *Palm OS Programmer's Companion*, vol. I for more information.

`prefDefaultAppCreator`	`UInt32`	9	Creator ID of the application that is launched after a reset. If 0, the system default application is launched.
`prefLocale`	`LmLocaleType`	9	The device's current locale, which specifies the country and language.
`prefDefFepPlugInCreator`	`UInt32`	10	Creator ID of the default FEP plug-in.
`prefColorThemeID`	`DmResID`	11	Resource ID of the color theme.
`prefHandednessChoice`	`UInt16`	12	Reserved for future use.
`prefHWRCreator`	`UInt32`	12	Creator ID of the handwriting recognition library.

Preferences in the User Interface

Table 48.2 shows the `SystemPreferencesChoice` constants and how they correspond to the values that users can set in the Preferences and Security applications. For further information about each preference, see Table 48.1.

Table 48.2 Preferences set in standard apps

Application/Panel	Field	SystemPreferencesChoice Constant
Preferences application General panel	Auto-off After	`prefAutoOffDuration`, `prefAutoOffDurationSecs` (Palm OS 3.5 and higher)
	Stay on in Cradle	`prefStayOnWhenPluggedIn`
	System Sound	`prefSysSoundLevelV20`, `prefSysSoundVolume`
	Alarm Sound	`prefAlarmSoundLevelV20`, `prefAlarmSoundVolume`
	Alarm Vibrate[1]	`prefAttentionFlags`
	Alarm LED[a]	`prefAttentionFlags`
	Game Sound	`prefGameSoundLevelV20`, `prefGameSoundVolume`
	Beam Receive field	`prefBeamReceive`
Preferences application Date & Time panel	Set Time Zone field	`prefTimeZone`
	Daylight Saving	`prefDaylightSaving Adjustment`

Table 48.2 Preferences set in standard apps *(continued)*

Application/Panel	Field	SystemPreferencesChoice Constant
Preferences application Formats panel	Preset to	`prefCountry`
	Time	`prefTimeFormat`
	Date	`prefDateFormat`, `prefLongDateFormat`
	Week starts	`prefWeekStartDay`
	Numbers	`prefNumberFormat`
Preferences application Buttons panel	Buttons on main panel	`prefHard1CharAppCreator`, `prefHard2CharAppCreator`, `prefHard3CharAppCreator`, `prefHard4CharAppCreator`, `prefCalcCharAppCreator`, `prefLauncherAppCreator`
	Pen button	`prefRonamaticChar`
	HotSync button	`prefHardCradleCharApp Creator` `prefHardCradle2CharApp Creator`
Security application	Current Privacy	`prefHidePrivateRecordsV33`, `prefShowPrivateRecords`
Security application	Lock & Turn Off button	`prefDeviceLocked`, `prefAutoLockType`, `prefAutoLockTime`, `prefAutoLockTimeFlag`

1. The Alarm Vibrate and Alarm LED preferences only appear on handhelds running Palm OS 4.0 and later that have the appropriate hardware capabilities. If you install Palm OS 4.0 on an older device, these preferences do not display.

Preferences Constants

The following table describes the constants defined in the `Preferences.h` header file.

Constant	Value	Description
`defaultAutoLockTime`	0	Initial setting for the `prefAutoLockTime` preference.
`defaultAutoLockTimeFlag`	0	Initial setting for the `prefAutoLockTimeFlag` preference.
`defaultAutoLockType`	never	Initial setting for the `prefAutoLockType` preference.
`defaultAutoOffDuration`	2	Initial setting for the `prefAutoOffDuration` preference, given in minutes.
`defaultAutoOffDuration Secs`	120	Initial setting for the `prefAutoOffDurationSecs` preference, given in seconds. The value is 2 times the number of seconds in a minute, or 120 seconds.
`defaultAlarmSoundLevel`	`slOn`	Initial setting for the `prefAlarmSoundLevelV20` preference.
`defaultAlarmSoundVolume`	`sndMaxAmp`	Initial setting for the `prefAlarmSoundVolume` preference.
`defaultGameSoundLevel`	`slOn`	Initial setting for the `prefGameSoundLevelV20` preference.
`defaultGameSoundVolume`	`sndMaxAmp`	Initial setting for the `prefGameSoundVolume` preference.

Constant	Value	Description
`defaultSysSoundLevel`	`slOn`	Initial setting for the `prefSysSoundLevelV20` preference.
`defaultSysSoundVolume`	`sndMaxAmp`	Initial setting for the `prefSysSoundVolume` preference.
`noPreferenceFound`	-1	The value that `PrefGetAppPreferences()` returns to indicate that preferences couldn't be found.
`preferenceDataVer2`	2	Version 2 of the system preferences structure, used for Palm OS 2.0.
`preferenceDataVer3`	3	Version 3 of the system preferences structure, used for Palm OS 3.0.
`preferenceDataVer4`	4	Version 4 of the system preferences structure, used for Palm OS 3.1.
`preferenceDataVer5`	5	Version 5 of the system preferences structure, used for Palm OS 3.2.
`preferenceDataVer6`	6	Version 6 of the system preferences structure, used for Palm OS 3.3.
`preferenceDataVer8`	8	Version 8 of the system preferences structure, used for Palm OS 3.5.
`preferenceDataVer9`	9	Version 9 of the system preferences structure, used for Palm OS 4.0.
`preferenceDataVer10`	10	Version 10 of the system preferences structure, used for Palm OS Garnet version 5.0.
`preferenceDataVer11`	11	Version 11 of the system preferences structure, used for Palm OS Garnet version 5.1.

Constant	Value	Description
`preferenceDataVer12`	12	Version 12 of the system preferences structure, used for Palm OS Garnet version 5.3SC.
`preferenceDataVerLatest`	`preferenceDataVer12`	The latest version of the system preferences structure.

Preferences Functions

PrefGetAppPreferences Function

Purpose Return a copy of an application's preferences resource.

Declared In `Preferences.h`

Prototype

```
Int16 PrefGetAppPreferences (UInt32 creator,
   UInt16 id, void *prefs, UInt16 *prefsSize,
   Boolean saved)
```

Parameters

→ *creator*
Creator ID of the application that owns the preferences.

→ *id*
ID number of the preferences resource to retrieve. The IDs `0x8000` through `0xFFFF` are reserved for system use.

← *prefs*
Pointer to a buffer to hold the preferences.

↔ *prefsSize*
Pointer to the size of the `prefs` buffer passed in. (Note that the pointer and the value to which it points *must* be initialized before you call `PrefGetApPreferences`.) Upon return, contains the number of bytes actually written or the number of bytes needed for the `prefs` structure.

To determine the required size for the `prefs` structure, set `prefsSize` to 0 and pass `NULL` for `prefs`. Upon return, the

`prefsSize` parameter contains the required size. Never set `prefs` to `NULL` without also setting `prefsSize` to 0.

Always compare the value returned in this parameter with the value you passed in. If the two values differ, you need to resize the `prefs` structure and call this function again.

→ *`saved`*
If `true`, retrieve the preferences from the "saved" preferences database, which is backed up during a HotSync operation. If `false`, retrieve the preferences from the "unsaved" preferences database, which is usually not backed up during a HotSync operation.

Returns Returns the version number of the retrieved preferences resource, or returns the constant `noPreferenceFound` if the preferences resource wasn't found. The returned version number is the same version number that was passed to the `PrefSetAppPreferences()` function.

Comments Use this function to retrieve the preferences that you previously set with the `PrefSetAppPreferences()` function. You typically call this function in your `StartApplication` function upon a normal launch. The values of the `id` and `saved` parameters should be the same as you specified when calling `PrefSetAppPreferences()`, and the `prefs` parameter should be a structure of the same type as you passed to `PrefSetAppPreferences()`. Most applications store all preferences in a single preferences resource retrieved by a single call to `PrefGetAppPreferences`, but this is not required. You can use multiple preferences resources if you wish.

Applications typically call `PrefGetAppPreferences` twice: once with `prefs` set to `NULL` and `prefsSize` set to 0 to determine how much memory to allocate for the preferences structure, and a second time after allocating the required amount of memory to obtain a copy of the application's preferences resource.

The version number returned by this function allows you to handle the case where a new version of the application is being run for the first time. You can compare the value returned by this function with the current version number to determine if you need to set default values for any preferences created by the current release. For more information, see the section "Updating Preferences Upon a New Release" on page 351 of the *Palm OS Programmer's Companion*, vol. I.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `PrefSetPreferences()`, `PrefGetAppPreferencesV10()`

PrefGetAppPreferencesV10 Function

Purpose Return a copy of an application's preferences.

Declared In `Preferences.h`

Prototype

```
Boolean PrefGetAppPreferencesV10 (UInt32 type,
   Int16 version, void *prefs, UInt16 prefsSize)
```

Parameters

→ *type*
Creator ID of the application that owns the preferences.

→ *version*
Version number of the application's preferences.

← *prefs*
Pointer to a buffer to hold preferences.

→ *prefsSize*
Size of the buffer passed.

Returns Returns `false` if the preference resource was not found or the preference resource contains the wrong version number.

Comments The content and format of an application preference is application-dependent.

Compatibility This function corresponds to the 1.0 version of `PrefGetAppPreferences`.

See Also `PrefSetPreferences()`, `PrefGetAppPreferences()`

PrefGetPreference Function

Purpose Return a system preference. Use this function instead of `PrefGetPreferences()`.

Declared In `Preferences.h`

Prototype

```
UInt32 PrefGetPreference
    (SystemPreferencesChoice choice)
```

Parameters → *choice*

A constant that specifies what preference to retrieve. See `SystemPreferencesChoice`.

Returns Returns the system preference or 0 if the preference could not be found. On debug ROMs, a non-fatal error message is also displayed if the specified preference cannot be found.

Comments This function replaces the 1.0 function `PrefGetPreferences()`. While `PrefGetPreferences` only lets you retrieve the whole system preferences structure, this function lets you specify which preference to retrieve.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `PrefSetPreference()`, `PrefGetAppPreferences()`, `PrefGetAppPreferencesV10()`

PrefGetPreferences Function

Purpose Return a copy of the system preferences.

Declared In `Preferences.h`

Prototype

```
void PrefGetPreferences (SystemPreferencesPtr p)
```

Parameters ← *p*

Pointer to system preferences.

Returns Returns nothing. Stores the system preferences in `p`.

Comments The `p` parameter points to a memory block allocated by the caller that is filled in by this function.

This function is often called in `StartApplication` to get localized settings.

NOTE: This function can only be used to retrieve preferences that were in the 1.0 version of the preferences structure.

See Also `PrefSetPreferences()`

PrefOpenPreferenceDB Function

Purpose Return a handle to either the saved or unsaved preference database.

Declared In `Preferences.h`

Prototype
```
DmOpenRef PrefOpenPreferenceDB (Boolean saved)
```

Parameters → *saved*

If `true`, open the "saved" preferences database, which is backed up during a HotSync operation. If `false`, open the "unsaved" preferences database, which usually is not backed up during a HotSync operation.

Returns Returns the handle, or 0 if an error results.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `PrefGetPreference()`, `PrefSetPreference()`, `PrefOpenPreferenceDBV10()`

PrefOpenPreferenceDBV10 Function

Purpose Return a handle to the system preference database.

Declared In `Preferences.h`

Prototype
```
DmOpenRef PrefOpenPreferenceDBV10 (void)
```

Parameters None.

Returns Returns the handle, or 0 if an error results.

Compatibility This function corresponds to the 1.0 version of `PrefOpenPreferenceDB`.

See Also `PrefGetPreferences()`, `PrefSetPreferences()`

PrefSetAppPreferences Function

Purpose Set an application's preferences in the specified preferences database.

Declared In `Preferences.h`

Prototype

```
void PrefSetAppPreferences (UInt32 creator,
   UInt16 id, Int16 version, const void *prefs,
   UInt16 prefsSize, Boolean saved)
```

Parameters

→ *creator*
Creator ID of the application that owns this preference.

→ *id*
ID number of the preference to set. An application can have multiple preferences. The IDs `0x8000` through `0xFFFF` are reserved for system use.

→ *version*
Version number of the application's preferences.

→ *prefs*
Pointer to a buffer that holds the current value of the preferences structure. Pass `NULL` if you want to delete the preferences.

→ *prefsSize*
Size of the buffer passed. Pass 0 if you want to delete the preferences structure.

→ *saved*
If `true`, saves the preferences in the "saved" preferences database. If not, saves the preferences in the "unsaved" preferences database.

Returns Returns nothing.

Comments You typically call this function in your `StopApplication` function to save the current state of the application or if the user has changed an application preference during the current session.

The "saved" preferences database is backed up when a user performs the HotSync operation. The "unsaved" preferences database is not backed up by default. (The user can use a third-party tool to set the backup bit in the "unsaved" preferences database, which would cause it to be backed up.) Both the "saved" and the "unsaved" preferences reside in the storage heap and thus persist

across soft resets. The only way that preferences are lost is if a hard reset is performed. "Which Preferences Database to Use" on page 349 of the *Palm OS Programmer's Companion*, vol. I describes how to choose between the "saved" and "unsaved" preferences databases.

The version number that you pass as the `version` parameter is returned when the preferences are retrieved by `PrefGetAppPreferences()`. You can use this version number to determine if a new release of the application is being run for the first time. For more information, see the section "Updating Preferences Upon a New Release" on page 351 of the *Palm OS Programmer's Companion*, vol. I.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `PrefSetAppPreferencesV10()`

PrefSetAppPreferencesV10 Function

Purpose Save an application's preferences in the preferences database.

Declared In `Preferences.h`

Prototype

```
void PrefSetAppPreferencesV10 (UInt32 creator,
    Int16 version, void *prefs, UInt16 prefsSize)
```

Parameters

→ *creator*
Creator ID of the application that owns this preference.

→ *version*
Version number of the application.

→ *prefs*
Pointer to a buffer holding preferences.

→ *prefsSize*
Size of the buffer passed.

Returns Returns nothing.

Comments The content and format of an application preference is application-dependent.

Compatibility This function corresponds to the 1.0 version of `PrefSetAppPreferences`.

See Also `PrefSetAppPreferences()`, `PrefGetPreferences()`

PrefSetPreference Function

Purpose Set a system preference. Using this function instead of `PrefSetPreferences` allows you to set selected preferences without having to access the whole structure.

Declared In `Preferences.h`

Prototype

```
void PrefSetPreference
    (SystemPreferencesChoice choice, UInt32 value)
```

Parameters

→ *choice*
A `SystemPreferencesChoice` constant specifying the preference to be set.

→ *value*
Value to assign to the preference.

Returns Returns nothing. If the specified preference cannot be found, displays a non-fatal error message on debug ROMs. On release ROMs, this function fails silently.

Compatibility Implemented only if 2.0 New Feature Set is present.

PrefSetPreferences Function

Purpose Set the system preferences.

Declared In `Preferences.h`

Prototype

```
void PrefSetPreferences (SystemPreferencesPtr p)
```

Parameters

→ *p*
Pointer to system preferences.

Returns Returns nothing.

Comments Unless there's a reason for you to access the whole preferences structure, call `PrefSetPreference()` instead.

NOTE: This function can only be used to set preferences that were in the 1.0 version of the preferences structure.

See Also `PrefGetPreferences()`

49

Rectangles

This chapter provides reference material for the rectangles API, declared in the header file `Rect.h`. It is divided into the following sections:

- Rectangle Data Structures
- Rectangle Functions

Rectangle Data Structures

PointType Struct

Purpose The `PointType` structure defines a point within a window or on the screen.

Prototype

```
typedef struct PointType {
  Coord x;
  Coord y;
} PointType;
```

Fields

`X`
: Horizontal coordinate.

`Y`
: Vertical coordinate.

RectangleType Struct

Purpose The `RectangleType` structure defines a rectangular portion of a window or of the screen.

Prototype

```
typedef struct RectangleType {
  PointType topLeft;
  PointType extent;
} RectangleType;
```

```
typedef RectangleType *RectanglePtr;
```

Fields

`topLeft`
: Coordinates of the upper-left corner of the rectangle relative to the window or screen in which the rectangle resides.

`extent`
: Width (`extent.x`) and height (`extent.y`) of the rectangle.

Rectangle Functions

RctCopyRectangle Function

Purpose Copy the source rectangle to the destination rectangle.

Declared In `Rect.h`

Prototype

```
void RctCopyRectangle
   (const RectangleType *srcRectP,
   RectangleType *dstRectP)
```

Parameters

`srcRectP`
: A pointer to the rectangle to be copied.

`dstRectP`
: A pointer to the destination rectangle.

See Also `RctSetRectangle()`

RctGetIntersection Function

Purpose Determine the intersection of two rectangles.

Declared In `Rect.h`

Prototype

```
void RctGetIntersection
   (const RectangleType *r1P,
   const RectangleType *r2P, RectangleType *r3P)
```

Parameters

`r1P`
: A pointer to a source rectangle.

`r2P`
: A pointer to the other source rectangle.

`r3P`
: Upon return, points to a rectangle representing the intersection of `r1` and `r2`.

Comments

The rectangle type `RectangleType`, which is pointed to by `RectanglePtr`, stores the coordinates for the top-left corner of the rectangle plus the rectangle's width and height. This function returns in the `r3` parameter a pointer to the rectangle that represents the intersection of the first two rectangles.

If the rectangles `r1` and `r2` do not intersect, `r3` contains a rectangle whose top-left coordinate is the maximum of `r1` and `r2`'s top-left coordinates and whose extent varies based on the location of the two rectangles.

Compatibility

On releases prior to Palm OS® 3.5, if rectangles `r1` and `r2` don't intersect, `r3` contains a rectangle that begins at coordinates (0,0 and has 0 width and 0 height. On Palm OS 3.5 and later, if the two rectangles don't intersect then `r3` contains a rectangle in which one or both of the extent coordinates is zero.

RctInsetRectangle Function

Purpose

Move all of the boundaries of a rectangle by a specified offset.

Declared In

`Rect.h`

Prototype

```
void RctInsetRectangle (RectangleType *rP,
    Coord insetAmt)
```

Parameters

`rP`
: A pointer to the rectangle.

`insetAmt`
: Number of pixels to move the boundaries. This can be a negative number.

Comments

The rectangle type `RectangleType`, which is pointed to by `RectanglePtr`, stores the coordinates for the top-left corner of the rectangle plus the rectangle's width and height. This function adds `insetAmt` to the x and y values of the top-left coordinate and then adjusts the width and the height accordingly so that all of the sides of the rectangle are contracted or expanded by the same amount.

A positive `insetAmt` creates a smaller rectangle that is contained inside the old rectangle's boundaries. A negative `insetAmt` creates a larger rectangle that surrounds the old rectangle.

See Also `RctOffsetRectangle()`

RctOffsetRectangle Function

Purpose Move the top and left boundaries of a rectangle by the specified values.

Declared In `Rect.h`

Prototype

```
void RctOffsetRectangle (RectangleType *rP,
    Coord deltaX, Coord deltaY)
```

Parameters

`rP`
: A pointer to the rectangle.

`deltaX`
: Number of pixels to move the left boundary. This can be a negative number.

`deltaY`
: Number of pixels to move the top boundary. This can be a negative number.

Comments The rectangle type `RectangleType`, which is pointed to by `RectanglePtr`, stores the coordinates for the top-left corner of the rectangle plus the rectangle's width and height. This function adds `deltaX` to the x value of the top-left coordinate and `deltaY` to the y value. The width and height are unchanged. Thus, this function shifts the position of the rectangle by the `deltaX` and `deltaY` amounts.

See Also `RctInsetRectangle()`

RctPtInRectangle Function

Purpose Determine if a point lies within a rectangle's boundaries.

Declared In `Rect.h`

Prototype

```
Boolean RctPtInRectangle (Coord x, Coord y,
   const RectangleType *rP)
```

Parameters

`x`
The x coordinate of the point.

`y`
The y coordinate of the point.

`rP`
The rectangle.

Returns Returns `true` if the point (`x`, `y`) lies within the boundaries of rectangle `r`, `false` otherwise.

RctSetRectangle Function

Purpose Sets a rectangle's values.

Declared In `Rect.h`

Prototype

```
void RctSetRectangle (RectangleType *rP,
   Coord left, Coord top, Coord width,
   Coord height)
```

Parameters

`rP`
A pointer to the rectangle to be set.

`left`
The x value for the top-left coordinate of the rectangle.

`top`
The y value for the top-left coordinate of the rectangle.

`width`
The rectangle's width.

`height`
The rectangle's height.

See Also `RctCopyRectangle()`

50

Sound Manager

This chapter describes the API that's defined by the Sound Manager. You use this API to make and record sounds.

The Sound Manager controls two independent sound facilities:

- "Simple" sound: single voice, monophonic, square-wave sound synthesis, useful for system beeps. This is the traditional (pre-Palm OS® Garnet) sound.
- "Sampled" sound: stereo, multi-format, sampled data recording and playback (new in Palm OS Garnet).

The organization of this chapter reflects this division:

- Simple Sound Data Structures
- Simple Sound Constants
- Simple Sound Functions
- Simple Sound Application-Defined Functions
- Sampled Sound Data Structures
- Sampled Sound Constants
- Sampled Sound Functions
- Sampled Sound Application-Defined Functions

All elements described here are declared in `SoundMgr.h`.

For more information on the Sound Manager, see the section "Sound" in the *Palm OS Programmer's Companion*, vol. I.

Simple Sound Data Structures

SndCallbackInfoType Struct

Purpose The `SndCallbackInfoType` structure encapsulates a callback function and its argument data. `SndCallbackInfoType` is used

by the SndSmfCallbacksType structure, which is used to list the callback functions that are called during SMF playback.

Prototype

```
typedef struct SndCallbackInfoType {
  MemPtr funcP;
  UInt32 dwUserData;
} SndCallbackInfoType;
```

Fields

`funcP`
: A pointer to the callback function.

`dwUserData`
: Data that's passed as an argument to the function.

SndCommandType Struct

Purpose

The `SndCommandType` structure encapsulates a sound synthesis operation and its associated parameters. It is used by the SndDoCmd() function.

Prototype

```
typedef struct SndCommandType {
  SndCmdIDType cmd;
  UInt8 reserved;
  Int32 param1;
  UInt16 param2;
  UInt16 param3;
} SndCommandType;
typedef SndCommandType *SndCommandPtr;
```

Fields

`cmd`
: Constant that represents a sound operation. The operations are listed and described in SndDoCmd().

`reserved`
: Reserved for future use.

`param1, param2, param3`
: Operation-specific parameters. The parameters' meanings are described in SndDoCmd().

Compatibility

The `reserved` field was introduced in the Palm OS 3.5 SDK. Consequently, when using a later SDK the compiler will complain when it encounters code that depends upon the presence or absence of this field. Code like this, for instance:

```
static SndCommandType onCmd = {sndCmdNoteOn,60,2200,127};
```

Instead of doing the above, set each of the structure's fields explicitly, like this:

```
SndCommandType onCmd;
[...]
onCmd.cmd = sndCmdNoteOn;
onCmd.param1 = 60;
onCmd.param2 = 2200;
onCmd.param3 = 127;
```

SndCmdIDType Typedef

Purpose Type used to specify a sound operation to `SndDoCmd()` The constants that can be assigned to this type are documented under "SndCmdIDTag" on page 933.

Prototype

```
typedef UInt8 SndCmdIDType;
```

SndMidiListItemType Struct

Purpose A structure that locates a MIDI file. This structure is used by the `SndCreateMidiList()` function.

Prototype

```
typedef struct SndMidiListItemType{
  Char name[sndMidiNameLength];
  UInt32 uniqueRecID;
  LocalID dbID;
  UInt16 cardNo;
} SndMidiListItemType;
```

Fields

`name`
: The null-terminated name of the MIDI file.

`uniqueRecID`
: The ID of the record that holds the MIDI file.

`dbID`
: Database ID of the database that holds the record.

`cardNo`
: Card number on which the database that holds the record resides.

SndMidiRecHdrType Struct

Purpose A structure that encapsulates the header of a MIDI record.

Prototype

```
typedef struct SndMidiRecHdrType {
  UInt32 signature;
  UInt8 bDataOffset;
  UInt8 reserved;
} SndMidiRecHdrType;
```

Fields

`signature`
: The MIDI record signature. Always set this field to `sndMidiRecSignature`.

`bDataOffset`
: Offset, in bytes, from the beginning of the record to the first byte of the MIDI data.

`reserved`
: Reserved for future use. Always set this field to 0.

SndMidiRecType Struct

Purpose The `SndMidiRecType` structure encapsulates the MIDI record header and the beginning of the human-readable name of the MIDI data.

Prototype

```
typedef struct SndMidiRecType {
  SndMidiRecHdrType header;
  Char name[2];
} SndMidiRecType
```

Fields

`header`
: The MIDI record header.

`name`
: The human-readable name of the MIDI data. The MIDI track name is a `null`-terminated string, even if it's empty. Contrary to the declaration, its length (including the null-

terminator) varies; it's at least one byte and at most `sndMidiNameLength` bytes long.

SndSmfCallbacksType Struct

Purpose A structure that contains a set of application-defined functions that are called during MIDI playback. To register your callback functions, call `SndPlaySmf()`.

Prototype

```
typedef struct SndSmfCallbacksType {
  SndCallbackInfoType completion;
  SndCallbackInfoType blocking;
  SndCallbackInfoType reserved;
} SndSmfCallbacksType;
```

Fields

`completion`
: Completion function; see `SndComplFuncType()`.

`blocking`
: Blocking function; see `SndBlockingFuncType()`.

`reserved`
: Reserved. Set this field to 0.

SndSmfChanRangeType Struct

Purpose A structure that defines the range of enabled MIDI channels. Events on MIDI channels outside the enabled range are ignored. By default, no channels are enabled.

Prototype

```
typedef struct SndSmfChanRangeType {
  UInt8 bFirstChan;
  UInt8 bLastChan;
} SndSmfChanRangeType;
```

Fields

`bFirstChan`
: The first enabled channel in the range [0, 15].

`bLastChan`
: The last enabled channel in the range [0, 15].

WARNING! The `SndSmfChanRangeType` structure expects MIDI channels to be in the range [0, 15]; real MIDI channel values are in the range [1, 16]. Thus, PalmSource MIDI channel 0 is real MIDI channel 1, PalmSource MIDI channel 1 is real MIDI channel 2, and so on.

SndSmfCmdEnum Typedef

Purpose Type used to specify a command to `SndPlaySmf()`. See "SndPlaySmf Commands" on page 935 for the set of values that this type can contain.

Prototype

```
typedef UInt8 SndSmfCmdEnum;
```

SndSmfOptionsType Struct

Purpose The `SndSmfOptionsType` structure establishes MIDI performance parameters. This structure is used with the `SndPlaySmf()` function to establish new parameter settings or to return the currently set values, depending on how the function is called. In the case where the structure returns values, only the "performance marker" fields (`dwStartMilliSec` and `dwEndMilliSec`) are valid.

Prototype

```
typedef struct SndSmfOptionsType {
  UInt32 dwStartMilliSec;
  UInt32 dwEndMilliSec;
  UInt16 amplitude;
  Boolean interruptible;
  UInt8 reserved1;
  UInt32 reserved;
} SndSmfOptionsType;
```

Fields

`dwStartMilliSec`

The "beginning of performance" marker, measured in milliseconds from the beginning of the track. A value of 0 plays the track from the beginning. The time difference between `dwStartMilliSec` and the performance time of the first subsequent MIDI event is respected. For example, if `dwStartMilliSec` is 2000 and the first (subsequent) note-

on event is at 3000, there will be a 1000 millisecond "pause" before the note is played.

`dwEndMilliSec`
: The "end of performance" marker, measured in milliseconds from the beginning of the track. To play to the end of the track, set this to `sndSmfPlayAllMilliSec`.

`amplitude`
: The volume of the track, in the range [0, `sndMaxAmp`]. The default is `sndMaxAmp`. If set to 0, the MIDI file isn't played.

`interruptible`
: If `true` (the default), MIDI playback is interrupted if the user interacts with the controls (digitizer, buttons, etc.), even if the interaction doesn't generate a sound command. If `false`, playback is not interrupted.

`reserved1`
: Reserved.

`reserved`
: Reserved. Set this field to 0.

SndSysBeepType Typedef

Purpose Type that specifies a system beep sound. This type is used by `SndPlaySystemSound()`. See "System Beeps" on page 935 for the set of values that this type can contain.

Prototype

```
typedef UInt8 SndSysBeepType;
```

Simple Sound Constants

SndCmdIDTag Enum

Purpose The `SndCmdIDTag` enumeration contains constants that represent specific sound operations used in simple sound playback with `SndDoCmd()`.

Prototype

```
typedef enum SndCmdIDTag {
  sndCmdFreqDurationAmp = 1,
  sndCmdNoteOn,
  sndCmdFrqOn,
  sndCmdQuiet
}
```

Constants

`sndCmdFreqDurationAmp`
: Play a tone. `SndDoCmd` blocks until the tone has finished.

`sndCmdFrqOn`
: Initiate a tone. `SndDoCmd` returns immediately while the tone plays in the background. Subsequent sound playback requests interrupt the tone.

`sndCmdNoteOn`
: Initiate a MIDI-defined tone. `SndDoCmd` returns immediately while the tone plays in the background. Subsequent sound playback requests interrupt the tone.

`sndCmdQuiet`
: Stop the playback of the currently generated tone.

Simple Sound Amplitudes

These constants can be supplied to the simple sound functions (such as `SndDoCmd()`) when an amplitude value is required. Note that these values are not compatible with the sampled sound amplitude range and thus shouldn't be used with the sampled sound functions.

Constant	Value	Description
sndMaxAmp	`64`	The maximum amplitude (full volume).
`sndDefaultAmp`	`sndMaxAmp`	The default amplitude.

sndMidiNameLength

Constant that defines the maximum string length, including the null terminator, for the name of a MIDI file or MIDI track.

```
#define sndMidiNameLength 32
```

sndMidiRecSignature

Constant that's used to tag a MIDI record. It is used as the value of the `signature` field of the `SndMidiRecHdrType` structure.

```
#define sndMidiRecSignature 'PMrc'
```

SndPlaySmf Commands Enum

Purpose This enumeration defines a set of commands that tell `SndPlaySmf()` whether it should play the file or simply return the duration of the file in milliseconds.

Prototype

```
enum SndSmfCmdEnumTag {
  sndSmfCmdPlay = 1,
  sndSmfCmdDuration,
}
```

Constants

`sndSmfCmdPlay`
: Play the specified Standard MIDI File.

`sndSmfCmdDuration`
: Return the duration, in milliseconds, of the specified Standard MIDI File.

sndSmfPlayAllMilliSec

Represents the (temporal) far end of a MIDI file. You can use this constant as the value of the `dwEndMilliSec` field of the `SndSmfOptionsType` structure before passing the structure to `SndPlaySmf()`. This setting tells the function to play the entire file.

```
#define sndSmfPlayAllMilliSec 0xFFFFFFFFUL
```

System Beeps Enum

Purpose This enumeration declares a set of constants that represent pre-defined system beeps. In order to play one of these sounds, pass the corresponding value to `SndPlaySystemSound()`.

Prototype

```
typedef enum SndSysBeepTag {
  sndInfo = 1,
  sndWarning,
  sndError,
  sndStartUp,
  sndAlarm,
  sndConfirmation,
  sndClick
}
```

Constants

`sndInfo`
: Heralds non-crucial information.

`sndWarning`
: Grabs the user's attention

`sndError`
: Indicates an illegal operation.

`sndStartUp`
: Played at device start up time.

`sndAlarm`
: Generic alarm sound; note that this is *not* the Datebook's alarm sound.

`sndConfirmation`
: Indicates approval or acceptance.

`sndClick`
: The button click sound.

Simple Sound Functions

SndCreateMidiList Function

Purpose Generate a list of MIDI records.

Declared In `SoundMgr.h`

Prototype

```
Boolean SndCreateMidiList (UInt32 creator,
   Boolean multipleDBs, UInt16 *recordCount,
   MemHandle *recordList)
```

Parameters

→ *creator*
Creator ID of the database in which the function looks for MIDI records. Pass `0` to search all databases.

→ *multipleDBs*
Pass `true` to search multiple databases for MIDI records. Pass `false` to search only in the first database that meets the search criteria.

← *recordCount*
Returns the number of MIDI records that were found.

← *recordList*
Returns a pointer to an array of `SndMidiListItemType` structures, one structure for each record that was found.

Returns Returns `true` if records were found, `false` otherwise.

Compatibility Implemented in Palm OS 3.0 and later.

SndDoCmd Function

Purpose Ask the Sound Manager to perform a simple sound synthesis operation.

Declared In `SoundMgr.h`

Prototype

```
Err SndDoCmd (void *channelP,
   SndCommandPtr commandP, Boolean noWait)
```

Parameters

→ *channelP*
Pointer to the sound channel on which you want to perform the operation. Pass `NULL` for the "shared" sound channel.

IMPORTANT: The Sound Manager only supports one channel of sound synthesis: You must pass NULL as the value of `channelP`.

→ *`commandP`*
: Pointer to a `SndCommandType` structure that describes the operation and contains any associated parameters. See the Comments section, below, for the set of sound commands and their associated parameters.

→ *`noWait`*
: Sets the function to be asynchronous (`true`) or synchronous (`false`) with respect to the caller.

IMPORTANT: `SndDoCmd` is always synchronous: The `noWait` value is currently ignored.

Returns Returns `errNone` if the operation completed successfully, or one of the following if an error occurred:

`sndErrBadParam`
: Invalid parameter.

`sndErrBadChannel`
: Invalid channel pointer.

`sndErrQFull`
: The sound queue is full.

Comments The sound operations that are performed by `SndDoCmd` are encapsulated in the `SndCommandType` structure. The `cmd` field represents the operation, while the `param` fields are data that's passed to the operation. The operations and data that `SndDoCmd` supports are described in the following table.

Table 50.1 SndDoCmd commands and parameters

Command	Function	Parameters
`sndCmdFreqDurationAmp`	Plays a tone. `SndDoCmd` blocks until the tone has finished.	`param1` is the tone's frequency in Hertz. `param2` is its duration in milliseconds `param3` is its amplitude in the range [0, `sndMaxAmp`]. If the amplitude is 0, the sound isn't played and the function returns immediately.
`sndCmdFrqOn`	Initiates a tone. `SndDoCmd` returns immediately while the tone plays in the background. Subsequent sound playback requests interrupt the tone.	`param1` is the tone's frequency in Hertz. `param2` is its duration in milliseconds `param3` is its amplitude in the range [0, `sndMaxAmp`]. If the amplitude is 0, the sound isn't played and the function returns immediately.

Table 50.1 SndDoCmd commands and parameters *(continued)*

Command	Function	Parameters
`sndCmdNoteOn`	Initiates a MIDI-defined tone. `SndDoCmd` returns immediately while the tone plays in the background. Subsequent sound playback requests interrupt the tone.	`param1` is the tone's pitch given as a MIDI key number in the range [0, 127]. `param2` is the tone's duration in milliseconds `param3` is its amplitude given as MIDI velocity [0, 127].
`sndCmdQuiet`	Stops the playback of the currently generated tone.	All parameter values are ignored.

Compatibility Commands other than `sndCmdFreqDurationAmp` are implemented in Palm OS 3.0 and later. In OS versions earlier than 3.0, `SndDoCmd` will crash with a fatal error if you pass a command other than `sndCmdFreqDurationAmp`.

See Also SndPlaySmf()

SndGetDefaultVolume Function

Purpose Return volume levels cached by the Sound Manager.

Declared In `SoundMgr.h`

Prototype

```
void SndGetDefaultVolume (UInt16 *alarmAmpP,
   UInt16 *sysAmpP, UInt16 *masterAmpP)
```

Parameters

← *alarmAmpP*
Pointer to the alarm amplitude.

← *sysAmpP*
Pointer to the system sound amplitude.

← *masterAmpP*
Pointer to the master amplitude.

Comments Pass `NULL` for those settings that you don't care about.

Compatibility Never call this function in Palm OS Garnet. To retrieve default volume levels, you should ask for the user's preferences settings.

See Also `SndSetDefaultVolume()`

SndInterruptSmfIrregardless Function

Purpose Stop the currently playing MIDI file, even if the performance was declared to be intolerant of interruptions.

Declared In `SoundMgr.h`

Prototype

```
Err SndInterruptSmfIrregardless (void)
```

Returns `Always returns errNone.`

Compatibility Implemented only if 4.0 New Feature Set is present. However, if Palm OS Cobalt Feature Set is present this function is unimplemented.

See Also SndPlaySmf(), SndPlaySmfResourceIrregardless()

SndPlaySmf Function

Purpose Perform a Standard MIDI File, or returns the duration of the file.

Declared In `SoundMgr.h`

Prototype

```
Err SndPlaySmf (void *channelP,
    SndSmfCmdEnum command, UInt8 *midiDataP,
    SndSmfOptionsType *optionsP,
    SndSmfChanRangeType *channelRangeP,
    SndSmfCallbacksType *callbacksP,
    Boolean noWait)
```

Parameters

→ *`channelP`*

A pointer to the sound channel on which you want to perform the MIDI file. Pass `NULL` for the "shared" sound channel.

IMPORTANT: The Sound Manager only supports one channel of sound synthesis: You must pass NULL as the value of `channel`.

→ *command*
: One of the SndPlaySmf Commands: either `SndSmfCmdPlay` (play the file) or `SndSmfCmdDuration` (return the duration of the file in milliseconds).

→ *midiDataP*
: The MIDI data; this can point to an `SndMidiRecHdrType` structure, or it can point directly to the actual MIDI data bytes in memory.

→ *optionsP*
: A pointer to a `SndSmfOptionsType` structure that defines performance parameters, such as volume, starting offset, and interruption tolerance. For default behavior, pass `NULL`. For more information, including default settings, see `SndSmfOptionsType`.

→ *channelRangeP*
: A pointer to a `SndSmfChanRangeType` structure that specifies the range of MIDI channels (in the SMF data) to use during playback. To play all channels, pass `NULL`.

→ *callbacksP*
: A pointer to a `SndSmfCallbacksType` structure that holds your callback functions. Pass `NULL` if you don't want any callbacks.

`noWait`
: This value is ignored. This function always finishes playing the SMF selection before returning (but see the Comments section, below).

Returns Returns `errNone` if the operation completed successfully, or one of the following if there was an error:

`sndErrBadParam`
: Invalid value passed to this function.

`sndErrBadChannel`
: Invalid sound channel.

`sndErrMemory`
: Insufficient memory.

`sndErrOpen`
: Tried to open channel that's already open.

`sndErrQFull`
: Can't accept more notes.

`sndErrFormat`
: Unsupported data format.

`sndErrBadStream`
: Invalid data stream.

`sndErrInterrupted`
: Play was interrupted.

Comments

Although this call is always synchronous, you can register a "blocking" function that's called periodically as the MIDI file is playing. See `SndBlockingFuncType()` for more information.

Normally, playback is halted by events generated by user interaction with the screen, digitizer, or hardware-based buttons. You can override this behavior by setting the `interruptible` field of the `optionsP` argument to `false`.

This function waits until any currently-playing simple sound has finished before starting playback of the requested MIDI data. A similar function, `SndPlaySmfIrregardless()`, doesn't wait: it interrupts the current performance and immediately begins playback of the requested data.

Compatibility

Implemented in Palm OS 3.0 and later.

SndPlaySmfIrregardless Function

Purpose Like SndPlaySmf(), but interrupts any currently playing simple sound, regardless of that sound's declared interruption tolerance.

Declared In SoundMgr.h

Prototype

```
Err SndPlaySmfIrregardless (void *channelP,
    SndSmfCmdEnum command, UInt8 *midiDataP,
    SndSmfOptionsType *optionsP,
    SndSmfChanRangeType *channelRangeP,
    SndSmfCallbacksType *callbacksP,
    Boolean noWait)
```

Parameters

→ *channelP*

A pointer to the sound channel on which you want to perform the MIDI file. Pass NULL for the "shared" sound channel.

IMPORTANT: The Sound Manager only supports one channel of sound synthesis: You must pass NULL as the value of channel.

→ *command*

One of the SndPlaySmf Commands: either SndSmfCmdPlay (play the file) or SndSmfCmdDuration (return the duration of the file in milliseconds).

→ *midiDataP*

The MIDI data; this can point to an SndMidiRecHdrType structure, or it can point directly to the actual MIDI data bytes in memory.

→ *optionsP*

A pointer to a SndSmfOptionsType structure that defines performance parameters, such as volume, starting offset, and interruption tolerance. For default behavior, pass NULL. For more information, including default settings, see SndSmfOptionsType.

→ *channelRangeP*

A pointer to a SndSmfChanRangeType structure that specifies the range of MIDI channels (in the SMF data) to use during playback. To play all channels, pass NULL.

→ *callbacksP*
: A pointer to a SndSmfCallbacksType structure that holds your callback functions. Pass NULL if you don't want any callbacks.

noWait
: This value is ignored. This function always finishes playing the SMF selection before returning.

Returns Returns errNone if the operation completed successfully, or one of the following if there was an error:

sndErrBadParam
: Invalid value passed to this function.

sndErrBadChannel
: Invalid sound channel.

sndErrMemory
: Insufficient memory.

sndErrOpen
: Tried to open channel that's already open.

sndErrQFull
: Can't accept more notes.

sndErrFormat
: Unsupported data format.

sndErrBadStream
: Invalid data stream.

sndErrInterrupted
: Play was interrupted.

Comments Although this call is always synchronous, you can register a "blocking" function that's called periodically as the MIDI file is playing. See SndBlockingFuncType() for more information.

Normally, playback is halted by events generated by user interaction with the screen, digitizer, or hardware-based buttons. You can override this behavior by setting the interruptible field of the optionsP argument to false.

This function doesn't wait for any currently-playing simple sound to finish before starting playback of the requested MIDI data: it interrupts the current performance and immediately begins playback of the requested data. A similar function, SndPlaySmf(),

waits for any currently-playing simple sound to finish before starting playback of the requested MIDI data.

Compatibility Implemented only if 4.0 New Feature Set is present. However, if Palm OS Cobalt Feature Set is present this function is unimplemented.

SndPlaySmfResource Function

Purpose Play a MIDI track read out of an open resource database.

Declared In `SoundMgr.h`

Prototype

```
Err SndPlaySmfResource (UInt32 resType,
    Int16 resID, SystemPreferencesChoice volume)
```

Parameters

→ *resType*
SMF resource type.

→ *resID*
SMF resource ID.

→ *volume*
Volume setting; one of `prefSysSoundVolume`, `prefGameSoundVolume`, or `prefAlarmSoundVolume` (all defined in `Preferences.h`).

Returns Returns `errNone` if the track was played successfully, or one of the following if an error occurred:

`sndErrBadParam`
The `volumeSelector` parameter is invalid or the SMF resource has invalid data.

`dmErrCantFind`
The specified resource doesn't exist.

`other values`
See SndPlaySmf()

Comments This function plays the entire MIDI file using all MIDI channels. Playback is interrupted by a key down or digitizer event. No callbacks are specified.

This function waits until any currently playing simple sound has finished before starting playback of the requested MIDI data. A similar function, `SndPlaySmfResourceIrregardless()`,

doesn't wait: It interrupts the current performance and immediately begins playback of the requested data.

Compatibility Implemented in Palm OS 3.2 and later.

SndPlaySmfResourceIrregardless Function

Purpose Like `SndPlaySmfResource()`, but interrupts any currently playing simple sound, regardless of that sound's declared tolerance for interruption.

Declared In `SoundMgr.h`

Prototype

```
Err SndPlaySmfResourceIrregardless (
   UInt32 resType, Int16 resID,
   SystemPreferencesChoice volumeSelector)
```

Parameters

→ *resType*
SMF resource type.

→ *resID*
SMF resource ID.

→ *volume*
Volume setting; one of `prefSysSoundVolume`, `prefGameSoundVolume`, or `prefAlarmSoundVolume`.

Returns Returns `errNone` if the track was played successfully, or one of the following if an error occurred:

`sndErrBadParam`
The `volumeSelector` parameter is invalid or the SMF resource has invalid data.

`dmErrCantFind`
The specified resource doesn't exist.

`other values`
See SndPlaySmf()

Comments This function plays the entire MIDI file using all MIDI channels. Playback is interrupted by a key down or digitizer event. No callbacks are specified.

This function doesn't wait for any currently playing simple sound to finish before starting playback of the requested MIDI data: it interrupts the current performance and immediately begins

playback of the requested data. A similar function, `SndPlaySmfResource()`, does wait.

Compatibility Implemented only if 4.0 New Feature Set is present. However, if Palm OS Cobalt Feature Set is present this function is unimplemented.

SndPlaySystemSound Function

Purpose Play a pre-defined (simple) system sound.

Declared In `SoundMgr.h`

Prototype

```
void SndPlaySystemSound (SndSysBeepType beepID)
```

Parameters

→ *beepID*
One of the system beep sound constants defined in the `System Beeps` enum.

Comments If you're playing an alarm (`sndAlarm`), the user's alarm volume preference setting is used. For all other system sounds, the system volume preference is used.

Alarm sounds (`sndAlarm`) are played synchronously: `SndPlaySystemSound` blocks until the sound has been played. All other sounds are played asynchronously.

SndSetDefaultVolume Function

Purpose Set the default sound volume levels cached by the Sound Manager.

Declared In `SoundMgr.h`

Prototype

```
void SndSetDefaultVolume (UInt16 *alarmAmpP,
    UInt16 *sysAmpP, UInt16 *masterAmpP)
```

Parameters

→ *alarmAmpP*
Pointer to the alarm amplitude.

→ *sysAmpP*
Pointer to the system sound amplitude.

→ *masterAmpP*
Pointer to the master amplitude.

Returns Returns nothing.

Comments Any of the arguments may be `NULL`. In that case, the corresponding setting is not altered.

NOTE: It is usually not appropriate for an application to be setting the default sound volume levels. Accordingly this function is rarely used by applications.

See Also `SndGetDefaultVolume()`

Simple Sound Application-Defined Functions

SndBlockingFuncType Function

Purpose A callback funtion that is invoked periodically during SMF playback.

Declared In `SoundMgr.h`

Prototype

```
Boolean SndBlockingFuncType (void *channelP,
    UInt32 userData, Int32 time)
```

Parameters

→ *`channelP`*
A pointer to the sound channel on which the file is being played. Currently always `NULL`.

→ *`userData`*
Application-defined data that's specified when the callback function is registered.

→ *`time`*
The amount of time, in milliseconds, available for completion of this function.

Returns Return `true` from your callback if playback is to continue. Return `false` if playback is to be aborted.

Comments Your application's blocking callback is called whenever the MIDI parser is "between notes." Your application can do whatever it wants during this period, as long as it doesn't take more than `time` milliseconds.

Specify your blocking callback function using the `blocking` field of the `SndSmfCallbacksType` structure that you pass to

SndPlaySmf(). Note that the `blocking` field is a SndCallbackInfoType structure; it contains a pointer to your callback function and a 32-bit value that is passed, as-is, to your callback.

See Also SndComplFuncType()

SndComplFuncType Function

Purpose A callback function that is invoked immediately after a MIDI file (SMF) finishes playing.

Declared In `SoundMgr.h`

Prototype

```
void SndComplFuncType (void *channelP,
    UInt32 userData)
```

Parameters

→ *channelP*
A pointer to the sound channel on which the file was played; currently always `NULL`.

→ *userData*
Application-defined data that's specified when the callback function is registered.

Comments Specify your blocking callback function using the `completion` field of the SndSmfCallbacksType structure that you pass to SndPlaySmf(). Note that the `completion` field is a SndCallbackInfoType structure; it contains a pointer to your callback function and a 32-bit value that is passed, as-is, to your callback.

Sampled Sound Data Structures

SndFormatType Typedef

Purpose Data type used to contain the encoding format of the data that you propose to pour through the stream with SndStreamCreateExtended(). See "Sound Data Encoding Formats" on page 952 for the set of values that this type can contain.

Prototype

```
typedef UInt32 SndFormatType;
```

SndPtr Typedef

Purpose Type used to cast a pointer to the sound data used by `SndPlayResource()`.

Prototype
```
typedef void *SndPtr;
```

Compatibility Implemented if Sound Stream Feature Set is present.

SndSampleType Typedef

Purpose Data type used to specify the sample format (size, data type, endianness) of a sampled sound stream. Used by `SndStreamCreate()`. See "SndSampleTag" on page 953 for the set of values that this type can contain.

Prototype
```
typedef Enum16 SndSampleType;
```

Compatibility Implemented if Sound Stream Feature Set is present.

SndStreamMode Typedef

Purpose Data type used to specify the "direction" (input or output) of a sampled sound stream. Used by the `SndStreamCreate()` function. See "SndStreamModeTag" on page 955 for the set of values that this type can contain.

Prototype
```
typedef Enum8 SndStreamMode;
```

Compatibility Implemented if Sound Stream Feature Set is present.

SndStreamRef Typedef

Purpose Data type that represents a sampled stream. You create an `SndStreamRef` with `SndStreamCreate()`.

Prototype
```
typedef UInt32 SndStreamRef;
```

Compatibility Implemented if Sound Stream Feature Set is present.

SndStreamWidth Typedef

Purpose Data type that specifies whether a sampled data stream is in stereo or mono. This data type is used by the `SndStreamCreate()` function. See "SndStreamWidthTag" on page 956 for the set of values that this type can contain.

Prototype

```
typedef Enum8 SndStreamWidth;
```

Compatibility Implemented if Sound Stream Feature Set is present.

Sampled Sound Constants

Sound Stream Feature Constants

To detect the presence of the Sound Stream Feature Set you check for the Sound Manager's version number feature. If the Sound Manager's version feature is defined, the Sound Stream feature set is supported.

Constant	Value	Description
`sndFtrIDVersion`	`0`	The feature number to supply to `FtrGet()`, along with a creator ID of `sysFileCSoundMgr`, when attempting to obtain the version of the Sound Manager.
`sndMgrVersionNum`	`(100)`	The current version of the Sound Manager.

Sound Data Encoding Formats Enum

Purpose The `SndFormatTag` enum defines a set of constants that represent various sound data encoding formats. Pass one of these constants as the `encodingFormat` argument to `SndStreamCreateExtended()`.

Prototype

```
enum SndFormatTag  {
  sndFormatPCM = 0,
  sndFormatIMA_ADPCM = 'APCM',
  sndFormatDVI_ADPCM = 'DPCM',
  sndFormatMP3 = 'MPG3',
  sndFormatAAC = 'DAAC',
  sndFormatOGG = 'OGGV'
}
```

Constants

`sndFormatPCM`
: Pulse Code Modulation format. This is the "no encoding" format; the data is a series of samples that are linear with regard to amplitude quantization and regular with regard to sampling rate.

`sndFormatIMA_ADPCM`
: The Interactive Multimedia Association's implementation of "adaptive delta" encoding. The sampling rate is constant, but the quantization is non-linear.

`sndFormatDVI_ADPCM`
: Microsoft's adaptive delta implementation. This is the same as IMA ADPCM.

`sndFormatMP3`
: Motion Picture Group Audio Layer III.

`sndFormatAAC`
: Dolby Advanced Audio Coding.

`sndFormatOGG`
: OGG Vorbis encoding.

Compatibility

Implemented if Sound Stream Feature Set is present. The Palm OS Garnet implementation of `SndStreamCreateExtended()` supports `sndFormatPCM` data only. To play ADPCM data, use `SndPlayResource()`.

SndSampleTag Enum

Purpose

The `SndSampleTag` enum defines a set of constants that represent the sample format (size, data type, endianness) of a sampled sound stream. These constant values are used with `SndStreamCreate()`.

The lower four bits of these constants gives the size (in bytes) of a single sample, as shown here:

```
UInt8 byteSize = formatConstant & 0x0f
```

Prototype

```
enum SndSampleTag {
  sndInt8 = 0x01,
  sndUInt8 = 0x11,
  sndInt16Big = 0x02,
  sndInt16Little = 0x12,
  sndInt32Big = 0x04,
  sndInt32Little = 0x14,
  sndFloatBig = 0x24,
  sndFloatLittle = 0x34,
#if CPU_ENDIAN == CPU_ENDIAN_BIG
  sndInt16 = sndInt16Big,
  sndInt16Opposite = sndInt16Little,
  sndInt32 = sndInt32Big,
  sndInt32Opposite = sndInt32Little,
  sndFloat = sndFloatBig,
  sndFloatOpposite= sndFloatLittle
#else
  sndInt16 = sndInt16Little,
  sndInt16Opposite = sndInt16Big,
  sndInt32 = sndInt32Little,
  sndInt32Opposite= sndInt32Big,
  sndFloat = sndFloatLittle,
  sndFloatOpposite= sndFloatBig
#endif
}
```

Constants

`sndFloat`
: Signed floating-point data in the device's native endianness.

`sndFloatBig`
: Signed floating-point data in big-endian format.

`sndFloatLittle`
: Signed floating-point data in little-endian format.

`sndFloatOpposite`
: Signed floating-point data in the endianness opposite to that of the device.

`sndInt16`
: Signed 16-bit integer data in the device's native endianness.

`sndInt16Big`
: Signed 16-bit integer data in big-endian format.

`sndInt16Little`
: Signed 16-bit integer data in little-endian format.

`sndInt16Opposite`
: Signed 16-bit integer data in the endianness opposite to that of the device.

`sndInt32`
: Signed 32-bit integer data in the device's native endianness.

`sndInt32Big`
: Signed 32-bit integer data in big-endian format.

`sndInt32Little`
: Signed 32-bit integer data in little-endian format.

`sndInt32Opposite`
: Signed 32-bit integer data in the endianness opposite to that of the device.

`sndInt8`
: Signed 8-bit data.

`sndUInt8`
: Unsigned 8-bit data.

Compatibility Implemented if Sound Stream Feature Set is present. In Palm OS Garnet, the 32-bit and floating point formats aren't supported.

SndStreamModeTag Enum

Purpose The `SndStreamModeTag` enum defines constants that represent the "direction" (input or output) of a sampled sound stream. Use these constants with the `SndStreamCreate()` function.

Prototype

```
enum SndStreamModeTag {
  sndInput,
  sndOutput
}
```

Constants

`sndInput`
Input stream used for recording.

`sndOutput`
Output stream used for playback.

Compatibility Implemented if Sound Stream Feature Set is present.

SndStreamWidthTag Enum

Purpose The `SndStreamWidthTag` defines constants that represent mono and stereo sampled data streams. Use these constants with the `SndStreamCreate()` function.

Prototype

```
enum SndStreamWidthTag {
  sndMono,
  sndStereo
}
```

Constants

`sndMono`
Mono (one channel) stream.

`sndStereo`
Stereo (two channel) stream.

Compatibility Implemented if Sound Stream Feature Set is present.

Sound Resource Playback Flags

Use these flags when calling `SndPlayResource()` to specify various settings. Currently, the only setting is function synchronization.

Constant	Value	Description
sndFlagSync	0x00000000	Tells `SndPlayResource` to wait until all sound data has been fed to the DAC before returning (meaning that the function will return just a bit before the sound has finished playing).

Constant	Value	Description
sndFlagAsync	0x00000001	Tells `SndPlayResource` to return immediately while playback continues in a separate thread.
sndFlagNormal	sndFlagSync	A shorthand for the set of "normal" flag settings.

Compatibility Implemented if Sound Stream Feature Set is present.

Stereo Pan Constants

These constants define the extremes and the midpoint when altering the stream's stereo balance with `SndStreamSetPan()`. `SndStreamSetPan` allows you to set the balance to one of these values, or any integral value between `sndPanFullLeft` and `sndPanFullRight`.

Constant	Value	Description
sndPanCenter	(0)	The stereo balance is centered.
sndPanFullLeft	(-1024)	The stereo balance is panned completely to the left.
sndPanFullRight	(1024)	The stereo balance is panned completely to the right.

Compatibility Implemented if Sound Stream Feature Set is present.

Volume Constants Enum

Purpose Use the volume constants defined in this enum with `SndStreamSetVolume()` and `SndPlayResource()`. The constants tell the functions to retrieve the named sound volume preference (as set by the user) and apply it as a volume setting.

Prototype

```
enum {
  sndSystemVolume = -1,
  sndGameVolume = -2,
  sndAlarmVolume = -3
}
```

Constants

`sndSystemVolume`
The user's system sound preference.

`sndGameVolume`
The user's game sound preference.

`sndAlarmVolume`
The uer's alarm sound preference.

Compatibility Implemented if Sound Stream Feature Set is present.

Sampled Sound Functions

SndPlayResource Function

Purpose Play formatted sound data read from a resource or file.

Declared In `SoundMgr.h`

Prototype

```
Err SndPlayResource (SndPtr soundP, Int32 volume,
    UInt32 flags)
```

Parameters

→ *soundP*
A pointer to the beginning of the formatted sound (including the header). Currently, only WAVE data is recognized (see the Comments section, below); in this case, `soundP` must point to the "RIFF" ID (byte 0 in a simple `.wav` file).

→ *volume*
Amplitude scalar, in the range [0, 32k]. See `SndStreamSetVolume()` for information on how amplitude scalar values are applied.

→ *flags*
One of the "Sound Resource Playback Flags" on page 956. Currently, the only setting is function synchronization: use `sndFlagSync` to have the function wait until all sound data has been fed to the DAC before returning , or `sndFlagAsync` flag to have the function return immediately while playback continues in a separate thread.

Returns Returns `errNone` if the operation completed successfully, or one of the following if there was an error:

`sndErrBadParam`
: The specified resource or file contains no data.

`sndErrFormat`
: The data is in an unsupported format.

`sndErrMemory`
: The function couldn't allocate sufficient memory.

`other errors`
: The device couldn't allocate system resources for the sound.

Comments

The supported WAVE parameters are:

- Uncompressed (PCM) or IMA 4-bit adaptive differential (IMA ADPCM). The ADPCM type is also known as DVI ADPCM; in a WAVE file, it's known as format 0x11.
- One or two-channels
- All normal sampling rates (8k, 11k, 22.05k, 44.1k, 48, 96k).

You can't interrupt or abort a resource playback once it's been initiated. The resource always plays to the end of the data.

Example

The following code excerpt shows how to use this function to play a sound resource.

```
SndPtr soundP;
MemHandle recordH;

recordH = DmGet1Resource(sysResTSound,TestWaveSound);
soundP = (SndPtr) MemHandleLock(recordH);
SndPlayResource(soundP,1024,sndFlagSync); // 1024 is 0dB
                                          // (unity) gain
MemHandleUnlock(recordH);
```

The above code first gets the resource from the last opened database. (You can use `DmGetResource()` instead if you have more than one database open.) It then locks the memory associated with the resource and converts the result to a pointer to a sournd resource.

The call to the `SndPlayResource()` function plays the sound. The second parameter is the sound level, which varies from 0 to 32767. A value of 1024 specifies unity gain. Higher values indicate higher gain. The third parameter specifies whether the function returns immediately or waits until after the sound has finished playing. You

should avoid using `sndFlagAsync`, which causes the function to return immediately without waiting for the sound to finish, if you use this code because you'll unlock the sound resource before the system finishes with it. In fact, you should always specify `sndFlagSync` unless you can guarantee two things:

1. Your resource memory remains locked for the duration of the sound
2. Your application does not exit for the duration of the sound

Remember that you cannot stop a sound played with `SndPlayResource()`.

Compatibility Implemented if Sound Stream Feature Set is present.

SndStreamCreate Function

Purpose Create a new audio data stream that can be used to record or play back uncompressed, sampled audio data.

Declared In `SoundMgr.h`

Prototype

```
Err SndStreamCreate (SndStreamRef *streamP,
    SndStreamMode mode, UInt32 sampleRate,
    SndSampleType type, SndStreamWidth width,
    SndStreamBufferCallback callback,
    void *callbackArgP, UInt32 bufferSize,
    Boolean callbackIsARM)
```

Parameters

← *streamP*
: Token that represents the newly created stream.

→ *mode*
: One of the SndStreamModeTag constants that represents the "direction" of the data stream. Either `sndInput` (for recording), or `sndOutput` (for playback).

→ *sampleRate*
: Sampling rate, in frames-per-second. Specify the native rate of the data, such as 22050, 44100, or 48000. The maximum rate is 96000.

→ *type*
Sample quantization and endianness (but see the section on "Data Formats," below). Supply one of the values documented under "SndSampleTag" on page 953.

→ *width*
One of the constants documented under "SndStreamWidthTag" on page 956 that represents the number of channels of data in the stream.

→ *callback*
A callback function that gets called when another buffer is needed. See `SndStreamBufferCallback()` for a description of the callback function that you must implement.

→ *callbackArgP*
Caller-defined data that gets passed to callback.

→ *bufferSize*
Preferred size (in frames) for the buffers that are passed to `callback`. Note that the actual buffer size (as allocated by the Sound Manager) may be different from this request.

→ *callbackIsARM*
Pass `true` if the callback function is written in ARM-native code, or `false` if it is written in 68k code.

Returns Returns `errNone` if the operation completed successfully, or one of the following if there was an error:

`sndErrBadParam`
`streamP` is invalid, `callback` is `NULL`, the sampling rate is too high (greater than 96000), or the device doesn't like some other sound parameter value.

`sndErrorMemory`
All streams are being used (there is a maximum of 16), or memory for this stream couldn't otherwise be allocated.

`other errors`
The device couldn't allocate system resources for the stream.

Comments This function creates a new audio stream into which you can write (playback) or from which you can read (record) buffers of uncompressed, sampled audio data. The stream's "direction"—

whether it will be used for recording or playback—is described by the `mode` argument.

You can create one input stream and as many as 15 output streams. The "active" end of a stream is hardwired to read from or write to the device's sound driver. This means you can't "redirect" an input stream to read from a file (for example), nor can you connect one output stream to another output stream in an attempt to create a filter chain. You can, however, collect data from the input stream, manipulate it, and then write it to an output stream.

Data Formats

The format of the data that flows through the stream is described by the `sampleRate`, `type`, and `width` arguments. If you're using an "extended" stream (see `SndStreamCreateExtended()`), you can also declare the data's encoding.

If you look at the SndSampleTag constants, you'll see four flavors for each quantization type: a big-endian version, a little-endian version, a native-endian version (defined as one of the other two), and an "opposite" version, which has endianness opposite that of the native version. In general, you should use the native-endian version when choosing a value for the `type` parameter. The one exception to this (in Palm OS Garnet) is if your application is written for 68k, but uses an ARM `callback` function (that is, `callbackIsARM == true`). In this case you should use one of the little-endian formats as the `type` value when you create your stream.

Running the Stream

The new stream starts running when you pass the `stream` token returned by this function to the `SndStreamStart()` function. This initiates a series of calls to your `callback` function, which is where the action is: Each `callback` invocation is passed a buffer into which you write or from which you read a chunk of audio data. The `callback` function is also passed the `callbackArg` that you supply here. See `SndStreamBufferCallback()` for more information on the callback function.

Buffering and Latency

Currently, audio streams are double-buffered. With regard to playback, this means that while one buffer (buffer A) is being played, your callback function is placing data in the other buffer (B). When A is "empty," the Sound Manager seamlessly starts playing buffer B, and passes buffer A back to your callback; when B is empty, it starts playing A, and passes back B, and so on. It's important that your `callback` function fills the data buffers as quickly as possible—certainly no longer than it takes to play a buffer of data. This same double-buffer scheme is also applied to sound recording although, of course, for recording you're emptying each buffer (and doing something with the data) in your `callback` function.

Regarding latency, you can use the `bufferSize` argument to suggest a buffer size and thereby increase or decrease latency, but you can't change the number of buffers. Keep in mind that the actual buffer size that's used may not be the same as the size you suggest; hardware and memory limitations may enforce a maximum or minimum buffer size. Also keep in mind that the `bufferSize` is measures in frames (not bytes).

See Also `SndStreamStart()`, `SndStreamDelete()`, `SndStreamBufferCallback()`

Compatibility Implemented if Sound Stream Feature Set is present.

SndStreamCreateExtended Function

Purpose Create a new audio data stream that can be used to record or play back audio data.

Declared In `SoundMgr.h`

Prototype

```
Err SndStreamCreate (SndStreamRef *streamP,
    SndStreamMode mode,
    SndFormatType encodingFormat,
    UInt32 sampleRate, SndSampleType type,
    SndStreamWidth width,
    SndStreamVariableBufferCallback callback,
    void *callbackArgP, UInt32 bufferSize,
    Boolean callbackIsARM)
```

Parameters

← *streamP*
Token that represents the newly created stream.

→ *mode*
One of the SndStreamModeTag constants that represents the "direction" of the data stream. Either `sndInput` (for recording), or `sndOutput` (for playback).

→ *encodingFormat*
Constant that represents the encoding format of the data that you propose to pour through the stream. See `SndFormatType` for a list of elligible values. Currently, only `sndFormatPCM` is supported.

→ *sampleRate*
Sampling rate, in frames-per-second. Specify the native rate of the data, such as 22050, 44100, or 48000. The maximum rate is 96000.

→ *type*
Sample quantization and endianness (see "Data Formats" on page 962 for advice on choosing this value). Supply one of the values documented under "SndSampleTag" on page 953.

→ *width*
One of the constants documented under "SndStreamWidthTag" on page 956 that represents the number of channels of data in the stream.

→ *callback*
A callback function that gets called when another buffer of data is needed. As implied by the name of the data type, the

function accepts variable-sized buffers. See `SndStreamVariableBufferCallback()` for a description of the callback function that you must implement.

→ *`callbackArgP`*
Caller-defined data that gets passed to callback.

→ *`bufferSize`*
Preferred size (in frames) for the buffers that are passed to `callback`. Note that the actual buffer size (as allocated by the Sound Manager) may be different from this request.

→ *`callbackIsARM`*
Pass `true` if the callback function is written in ARM-native code, or `false` if it is written in 68k code.

Returns Returns `errNone` if the operation completed successfully, or one of the following if there was an error:

`sndErrBadParam`
`streamP` is invalid, `callback` is `NULL`, the sampling rate is too high (greater than 96000), or the device doesn't like some other sound parameter value.

`sndErrorMemory`
All streams are being used (there is a maximum of 16), or memory for this stream couldn't otherwise be allocated.

`other errors`
The device couldn't allocate system resources for the stream.

Comments With a few minor exceptions, this function is equivalent to `SndStreamCreate()`; see that function's Comments section for a description of how the stream creation functions generally work.

One difference between standard and extended streams: If you're using an extended stream, you can also declare the data's encoding. When selecting that encoding, be aware of the following:

- The data format that you specify for an input stream must match the data that's produced by the audio hardware.
- For an output stream, you can specify any of the formats that the Sound Manager supports; the data is automatically converted to the output hardware's native audio format. Whether your stream's format setting actually affects the hardware is undefined. For example, if you set an output

stream to use a 48k sampling rate, that doesn't mean that the DAC will be set to 48k.

- Currently, only `sndFormatPCM` is supported.

Extended streams also allow for variable-sized buffers, as opposed to the fixed-sized buffers used by `SndStreamCreate`.This enables support for variable-length encoded data (such as MP3). To accommodate the variable-sized buffer, the callback function's prototype changes slightly for an extended stream: see `SndStreamVariableBufferCallback()` for a full description of the callback function you use with `SndStreamCreateExtended`.

SndStreamDelete Function

Purpose Stop the stream and destroy it.

Declared In `SoundMgr.h`

Prototype

```
Err SndStreamDelete (SndStreamRef stream)
```

Parameters

→ *stream*
Stream token, as returned from `SndStreamCreate()` or `SndStreamCreateExtended()`.

Returns Returns `errNone` if the operation completed successfully. Returns `sndErrBadParam` if the `stream` argument is invalid.

Comments `SndStreamDelete` calls `SndStreamStop()` before destroying the stream. You should never call `SndStreamDelete` from within your callback function.

Compatibility Implemented if Sound Stream Feature Set is present.

SndStreamGetPan Function

Purpose Retrieve a stream's stereo balance.

Declared In `SoundMgr.h`

Prototype

```
Err SndStreamGetPan (SndStreamRef stream,
    Int32 *panP)
```

Parameters

→ *stream*
Stream token, as returned from `SndStreamCreate()` or `SndStreamCreateExtended()`.

← *panP*
Pan value in the range [-1024 (hard left), 1024 (hard right)]. Center balance is 0.

Returns Returns `errNone` if the operation completed successfully. Returns `sndErrBadParam` if `stream` is invalid or `panP` is `NULL`.

Compatibility Implemented if Sound Stream Feature Set is present.

See Also `SndStreamSetPan()`

SndStreamGetVolume Function

Purpose Retrieve the amplitude scalar for a sound stream.

Declared In `SoundMgr.h`

Prototype

```
Err SndStreamGetVolume (SndStreamRef stream,
    Int32 *volumeP)
```

Parameters

→ *stream*
Stream token, as returned from `SndStreamCreate()` or `SndStreamCreateExtended()`.

← *volumeP*
Amplitude scalar, in the range [0, 32k]. See `SndStreamSetVolume()` for more information.

Returns Returns `errNone` if the operation completed successfully. Returns `sndErrBadParam` if `stream` is invalid or `volumeP` is `NULL`.

See Also `SndStreamSetVolume()`

Compatibility Implemented if Sound Stream Feature Set is present.

SndStreamPause Function

Purpose Pause or resume a sample stream.

Declared In `SoundMgr.h`

Prototype

```
Err SndStreamPause (SndStreamRef stream,
    Boolean pause)
```

Parameters

→ *stream*
Stream token, as returned from `SndStreamCreate()` or `SndStreamCreateExtended()`.

→ *pause*
If `true`, the function pauses the stream; if `false`, it resumes the stream

Returns Returns `errNone` if the operation completed successfully (which includes the situation where the stream is already in the requested state). Returns `sndErrBadParam` if `stream` is invalid.

Comments Currently, `SndStreamPause` simply calls `SndStreamStop()` (if `pause` is `true`) or `SndStreamStart()` (if `pause` is `false`). See those functions for details about "pausing" and "resuming" a sound stream.

You can't nest pauses; a single resume request is effective, regardless of the number of times the stream has been told to pause.

Compatibility Implemented if Sound Stream Feature Set is present.

SndStreamSetPan Function

Purpose Sets a stream's stereo balance.

Declared In `SoundMgr.h`

Prototype

```
Err SndStreamSetPan (SndStreamRef stream,
    Int32 pan)
```

Parameters

→ *stream*
Stream token, as returned from `SndStreamCreate()` or `SndStreamCreateExtended()`.

→ *pan*
Pan value in the range [-1024 (full left), 1024 (full right)]. Center balance is 0. As a convenience, you can use the values described in "Stereo Pan Constants." Note that values

outside of the valid range may yield unexpected results (but don't generate an error).

Returns Returns `errNone` if the operation completed successfully. Returns `sndErrBadParam` if `stream` is invalid.

Comments The pan value is used as a scalar on a channel's volume such that a channel increases from 0 (inaudible) to full volume as the pan value moves from an extreme to 0. Graphically, it looks like this:

Compatibility Implemented if Sound Stream Feature Set is present.

See Also `SndStreamGetPan()`

SndStreamSetVolume Function

Purpose Sets the amplitude scalar for a sound stream.

Declared In `SoundMgr.h`

Prototype

```
Err SndStreamSetVolume (SndStreamRef stream,
    Int32 volume)
```

Parameters

→ *stream*
Stream token, as returned from `SndStreamCreate()` or `SndStreamCreateExtended()`.

→ *volume*
Amplitude scalar in the range [0, 32k]. Values less than 0 are converted to 1024 (unity gain).

Returns Returns `errNone` if the operation completed successfully. Returns `sndErrBadParam` if `stream` is invalid.

Comments The `volume` value is applied as an amplitude scalar on the samples that this stream's callback function produces. The scalar is in the range [0, 32k], where 1024 is unity gain (i.e. the samples are multiplied by 1.0). The mapping of `volume` to a scalar is linear; thus a `volume` of 512 scales the samples by ~.5, 2048 scales by ~2.0, and so on.

To specify a user preference volume setting, supply one of the constants documented under "Volume Constants" on page 957. These values are guaranteed to be less than unity gain.

If the stream is stereo, both channels are scaled by the same amplitude scalar. To adjust the balance between the channels, use `SndStreamSetPan()`.

Compatibility Implemented if Sound Stream Feature Set is present.

SndStreamStart Function

Purpose Start a sample stream running.

Declared In `SoundMgr.h`

Prototype

```
Err SndStreamStart (SndStreamRef stream)
```

Parameters → *stream*

Stream token, as returned from `SndStreamCreate()` or `SndStreamCreateExtended()`.

Returns Returns `errNone` if the operation completed successfully (`errNone` is returned even if the stream is already running). Returns `sndErrBadParam` if `stream` is invalid.

Comments If the stream is already running, `SndStreamStart` returns immediately (with `errNone`). If it isn't running, the function starts the stream by initiating invocations of its callback function. If the stream is paused (through `SndStreamPause()`), the stream is resumed.

You can call this function from within another stream's callback function. This allows one stream to tell another stream to start playing.

Compatibility Implemented if Sound Stream Feature Set is present.

SndStreamStop Function

Purpose Stop a sample stream from running.

Declared In `SoundMgr.h`

Prototype

```
Err SndStreamStop (SndStreamRef stream)
```

Parameters

→ *stream*
Stream token, as returned from `SndStreamCreate()` or `SndStreamCreateExtended()`.

Returns Returns `errNone` if the operation completed successfully (`errNone` is returned even if the stream is already stopped). Returns `sndErrBadParam` if `stream` is invalid.

Comments Stops a running sound stream by neglecting to call the stream's callback function. The stream remains in this suspended state until you call `SndStreamStart()`.

You can call this function from the stream's own callback function. In other words, a stream can stop itself.

Compatibility Implemented if Sound Stream Feature Set is present.

Sampled Sound Application-Defined Functions

SndStreamBufferCallback Function

Purpose In input mode, delivers a data buffer to your application. In output mode, allows your application to supply the next buffer's worth of data.

Declared In `SoundMgr.h`

Prototype

```
Err (*SndStreamBufferCallback) (void *userDataP,
    SndStreamRef stream, void *bufferP,
    UInt32 frameCount)
```

Parameters

→ *userDataP*
Caller-defined data, as provided in the `callbackArg` parameter to `SndStreamCreate()`.

→ *stream*
Token that represents the stream to which this buffer belongs.

→ *bufferP*
The data buffer.

→ *frameCount*
Number of sample frames the buffer contains.

Returns Currently, the return value is ignored.

Comments The `SndStreamBufferCallback` function that you create is invoked in input (recording) mode when the Sound Manager wants to deliver a new buffer of sound data to your application. In output (playback) mode, it is invoked when the Sound Manager needs another buffer of sound data. You associate your callback function with a given stream when you call `SndStreamCreate()`.

In input mode, your callback function should read the data from the data buffer. In output mode, your callback function should write data into the data buffer (and it must fill the entire buffer with data).In either case, you want to do this as quickly as possible to avoid data underflow.

Note that the arguments passed to your callback function tell you nothing about the format of the data. You can use the `userDataP` argument to pass this information into the function.

The callback function is executed in a task that's created and managed by the Sound Manager. Because of this, the function doesn't have access to the symbols that you create in your application. Again, use the `userDataP` argument if you need to pass pointers to your symbols.

Compatibility Implemented if Sound Stream Feature Set is present.

See Also `SndStreamCreate()`, `SndStreamVariableBufferCallback()`

SndStreamVariableBufferCallback Function

Purpose In input mode, delivers a variable-length data buffer to your application. In output mode, allows your application to supply the next buffer's worth of data.

Declared In `SoundMgr.h`

Prototype

```
Err (*SndStreamVariableBufferCallback) (
    void *userDataP, SndStreamRef stream,
    void *bufferP, UInt32 *bufferSizeP)
```

Parameters

→ *`userDataP`*
Caller-defined data, as provided by the `callbackArg` parameter to `SndStreamCreateExtended()`.

→ *`stream`*
Token that represents the stream that this buffer belongs to.

→ *`bufferP`*
The data buffer.

↔ *`bufferSizeP`*
Size of the buffer, in bytes.

Returns Currently, the return value is ignored.

Comments The `SndStreamVariableBufferCallback` function that you create is invoked in input (recording) mode when the Sound Manager wants to deliver a new buffer of sound data to your application. In output (playback) mode, it is invoked when the Sound Manager needs another buffer of sound data. You associate your callback function with a given stream when you call `SndStreamCreateExtended()`.

In input mode, your callback function should read the data from the data buffer. In output mode, your callback function should write data into the data buffer and then reset `*bufferSizeP` to the amount of data that was actually written. Unlike `SndStreamBufferCallback()`, your callback function is not required to fill the entire buffer with data. Moreover, the data that it writes to the buffer doesn't have to meet any other threshold or requirement—for example, the buffer doesn't have to represent a certain amount of playback time. This flexibility is provided in order to support variable-length encoded data (such as MP3).

Whether your callback is reading from the buffer or writing to it, it should do this as quickly as possible to avoid data underflow.

Note that the arguments passed to your callback function tell you nothing about the format of the data. You can use the `userDataP` argument to pass this information into the function.

The callback function is executed in a task that's created and managed by the Sound Manager. Because of this, the function doesn't have access to the symbols that you create in your application. Again, use the `userDataP` argument if you need to pass pointers to your symbols.

Compatibility Implemented if Sound Stream Feature Set is present.

See Also `SndStreamCreateExtended()`, `SndStreamBufferCallback()`

51

Standard IO

This chapter provides reference material for the standard IO API:

- Standard IO Functions and Macros
- Standard IO Provider Functions
- Application-Defined Function

The header files `StdIOPalm.h` and `StdIOProvider.h` declare the standard IO API. For more information on using the standard IO API, see the chapter "Standard IO Applications"in the *Palm OS Programmer's Companion*, vol. I.

Standard IO Functions and Macros

The macros and functions in this section enable standard IO.

fgetc Macro

Purpose Calls `Siofgetc()` to return the next character from the input stream.

Declared In `StdIOPalm.h`

Prototype

```
#define fgetc (fs)
```

Parameters

→ *fs*

An input stream from which to read the next character. You can specify only the value `stdin` for this parameter; alternate streams are not currently implemented.

Returns The next character from the input stream. The return value `EOF` indicates an error occurred.

fgets Macro

Purpose Calls `Siofgets()` to return a string from the input stream.

Declared In `StdIOPalm.h`

Prototype

```
#define fgets (strP, maxChars, fs)
```

Parameters

← *strP*
: A pointer to the returned string.

→ *maxChars*
: The number of characters to read from the input stream, plus one for the null terminator.

→ *fs*
: An input stream from which to read the next character. You can specify only the value `stdin` for this parameter; alternate streams are not currently implemented.

Returns A pointer to the string read from the input stream. If an error or EOF occurs before any characters are read, returns `NULL`.

Comments The returned string is always terminated by a null character.

fprintf Macro

Purpose Calls `Siofprintf()` to write formatted output to an output stream.

Declared In `StdIOPalm.h`

Prototype

```
#define fprintf (fs, formatP, ...)
```

Parameters

→ *fs*
: An output stream to which to write the formatted output. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

→ *formatP*
: A pointer to a format string that controls how subsequent arguments are converted for output.

→ *...*
: Zero or more parameters to be formatted as specified by the `formatP` string.

Returns Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

Comments This function internally calls `StrVPrintF()` to do the formatting. See that function for details on which format specifications are supported.

fputc Macro

Purpose Calls `Siofputc()` to write a character to the output stream.

Declared In `StdIOPalm.h`

Prototype

```
#define fputc (c, fs)
```

Parameters

→ *c*
A character to write to the output stream.

→ *fs*
An output stream to which to write the character. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

Returns The character that was written. If an error occurs, the value `EOF` is returned.

fputs Macro

Purpose Calls `Siofputs()` to write a string to the output stream.

Declared In `StdIOPalm.h`

Prototype

```
#define fputs (strP, fs)
```

Parameters

→ *strP*
A pointer to the string to write.

→ *fs*
An output stream to which to write the string. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

Returns Returns 0 on success and the value `EOF` on error.

getchar Macro

Purpose Calls `Siofgetc()` to read the next character from the `stdin` input stream.

Declared In `StdIOPalm.h`

Prototype

```
#define getchar ()
```

Returns The next character from the input stream. The return value `EOF` indicates an error occurred.

gets Macro

Purpose Calls `Siogets()` to read a string from the `stdin` input stream.

Declared In `StdIOPalm.h`

Prototype

```
#define gets (strP)
```

Parameters

← *strP*
A pointer to the returned string.

Returns A pointer to the string read from the input stream. If an error or EOF occurs before any characters are read, returns `NULL`.

Comments The returned string does not include a null terminator. You must ensure that the input line, if any, is sufficiently short to fit in the string.

printf Macro

Purpose Calls `Sioprintf()` to write formatted output to the stdout output stream.

Declared In `StdIOPalm.h`

Prototype

```
#define printf (formatP, ...)
```

Parameters

→ *formatP*
A pointer to a format string that controls how subsequent arguments are converted for output.

→ `...`
Zero or more parameters to be formatted as specified by the `formatP` string.

Returns Returns the number of characters written out (not including the null terminator used to end output strings).

Comments This function internally calls `StrVPrintF()` to do the formatting. See that function for details on which format specifications are supported. Returns a negative number if there is an error.

putc Macro

Purpose Calls `Siofputc()` to write a character to the output stream.

Declared In `StdIOPalm.h`

Prototype

```
#define putc (c, fs)
```

Parameters

→ *c*
A character to write to the output stream.

→ *fs*
An output stream to which to write the character. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

Returns The character that was written. If an error occurs, the value `EOF` is returned.

putchar Macro

Purpose Calls `Siofputc()` to write a character to the `stdout` output stream.

Declared In `StdIOPalm.h`

Prototype

```
#define putchar (c)
```

Parameters

→ *c*
A character to write to the `stdout` output stream.

Returns The character that was written. If an error occurs, the value `EOF` is returned.

puts Macro

Purpose Calls `Sioputs()` to write a string to the output stream `stdout`.

Declared In `StdIOPalm.h`

Prototype

```
#define puts (strP)
```

Parameters → *strP*
A pointer to the string to write to `stdout`.

Returns Returns a nonnegative value on success and the value `EOF` on error.

SioAddCommand Function

Purpose Adds a built-in command that is supplied by the standard IO provider application.

Declared In `StdIOPalm.h`

Prototype

```
void SioAddCommand (const Char *cmdStr,
    SioMainProcPtr cmdProcP)
```

Parameters → *cmdStr*
Pointer to a string that is the command name.

→ *cmdProcP*
Pointer to the command entry point function (the `SioMain()` function).

Returns Returns nothing.

Comments This routine is useful for registering a command that is inside the standard IO provider application instead of in its own database.

This routine must be used to test commands under the Simulator since it can't launch application databases.

Siofgetc Function

Purpose Return the next character from the input stream.

Declared In `StdIOPalm.h`

Prototype

```
Int16 Siofgetc (FILE *fs)
```

Parameters

→ *fs*

An input stream from which to read the next character. You can specify only the value `stdin` for this parameter; alternate streams are not currently implemented.

Returns The next character from the input stream. The return value `EOF` indicates an error occurred.

See Also `fgetc()`

Siofgets Function

Purpose Return a string from the input stream.

Declared In `StdIOPalm.h`

Prototype

```
Char *Siofgets (Char *strP, UInt16 maxChars,
   FILE *fs)
```

Parameters

← *strP*

A pointer to the returned string.

→ *maxChars*

The number of characters to read from the input stream, plus one for the null terminator.

→ *fs*

An input stream from which to read the next character. You can specify only the value `stdin` for this parameter; alternate streams are not currently implemented.

Returns A pointer to the string read from the input stream. If an error or EOF occurs before any characters are read, returns `NULL`.

Comments The returned string is always terminated by a null character.

See Also `fgets()`

Siofprintf Function

Purpose Write formatted output to an output stream.

Declared In `StdIOPalm.h`

Prototype

```
Int16 Siofprintf (FILE *fs, const Char *formatP,
    ...)
```

Parameters

→ *fs*
An output stream to which to write the formatted output. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

→ *formatP*
A pointer to a format string that controls how subsequent arguments are converted for output.

→ *...*
Zero or more parameters to be formatted as specified by the `formatP` string.

Returns Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

Comments This function internally calls `StrVPrintF()` to do the formatting. See that function for details on which format specifications are supported.

See Also `fprintf()`

Siofputc Function

Purpose Write a character to the output stream.

Declared In `StdIOPalm.h`

Prototype

```
Int16 Siofputc (Int16 c, FILE *fs)
```

Parameters

→ *c*
A character to write to the output stream.

→ *fs*
An output stream to which to write the character. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

Returns The character that was written. If an error occurs, the value `EOF` is returned.

See Also `fputc()`

Siofputs Function

Purpose Write a string to the output stream.

Declared In `StdIOPalm.h`

Prototype

```
Int16 Siofputs (const Char *strP, FILE *fs)
```

Parameters

→ *strP*
A pointer to the string to write.

→ *fs*
An output stream to which to write the string. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

Returns Returns 0 on success and the value `EOF` on error.

See Also `fputs()`

Siogets Function

Purpose Read a string from the `stdin` input stream.

Declared In `StdIOPalm.h`

Prototype

```
Char *Siogets (Char *strP)
```

Parameters

← *strP*
A pointer to the returned string.

Returns A pointer to the string read from the input stream. If an error or EOF occurs before any characters are read, returns `NULL`.

Comments The returned string does not include a null terminator. You must ensure that the input line, if any, is sufficiently short to fit in the string.

See Also `gets()`

Sioprintf Function

Purpose Write formatted output to the `stdout` output stream.

Declared In `StdIOPalm.h`

Prototype

```
Int16 Sioprintf (const Char *formatP, ...)
```

Parameters

→ *`formatP`*
A pointer to a format string that controls how subsequent arguments are converted for output.

→ `...`
Zero or more parameters to be formatted as specified by the `formatP` string.

Returns Returns the number of characters written out (not including the null terminator used to end output strings).

Comments This function internally calls `StrVPrintF()` to do the formatting. See that function for details on which format specifications are supported. Returns a negative number if there is an error.

See Also `printf()`

Sioputs Function

Purpose Write a string to the output stream `stdout`.

Declared In `StdIOPalm.h`

Prototype

```
Int16 Sioputs (const Char *strP)
```

Parameters

→ *`strP`*
A pointer to the string to write to `stdout`.

Returns Returns a nonnegative value on success and the value `EOF` on error.

See Also `puts()`

Siosystem Function

Purpose Execute another Stdio command.

Declared In `StdIOPalm.h`

Prototype

```
Int16 Siosystem (const Char *cmdStrP)
```

Parameters

→ *cmdStrP*
A pointer to a string containing the command line to execute.

Returns Returns a value >= 0 on success or < 0 on failure.

Comments This function first looks for a built-in command with the specified name. If none is found, it looks for a Stdio application database with the name "Cmd-*cmdname*" where *cmdname* is the first word in the command string `cmdStrP`.

See Also `SioExecCommand()`, `system()`

Siovfprintf Function

Purpose Write formatted output to the `stdout` output stream.

Declared In `StdIOPalm.h`

Prototype

```
Int16 Siovfprintf (FILE *fs, const Char *formatP,
    _Palm_va_list args)
```

Parameters

→ *fs*
An output stream to which to write the formatted output. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

→ *formatP*
A pointer to a format string that controls how subsequent arguments are converted for output.

→ *args*
A pointer to a list of zero or more parameters to be formatted as specified by the `formatP` string.

Returns Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

Comments This function internally calls `StrVPrintF()` to do the formatting. See that function for details on which format specifications are supported.

See Also `vfprintf()`

sprintf Macro

Purpose Calls `StrPrintF()` to write formatted output to the stdout output stream.

Declared In `StdIOPalm.h`

Prototype

```
#define sprintf (formatP, ...)
```

Parameters

→ *formatP*
A pointer to a format string that controls how subsequent arguments are converted for output.

→ `...`
Zero or more parameters to be formatted as specified by the `formatP` string.

Returns Returns the number of characters written out (not including the null terminator used to end output strings).

Comments See `StrVPrintF()` for details on which format specifications are supported. Returns a negative number if there is an error.

system Macro

Purpose Calls `Siosystem()` to execute another Stdio command.

Declared In `StdIOPalm.h`

Prototype

```
#define system (cmdStrP)
```

Parameters

→ *cmdStrP*
A pointer to a string containing the command line to execute.

Returns Returns a value >= 0 on success or < 0 on failure.

Comments This function first looks for a built-in command with the specified name. If none is found, it looks for a Stdio application database with

the name "Cmd-*cmdname*" where *cmdname* is the first word in the command string `cmdStrP`.

See Also `SioExecCommand()`

vfprintf Macro

Purpose Calls `Siovfprintf()` to write formatted output to the stdout output stream.

Declared In `StdIOPalm.h`

Prototype

```
#define vfprintf (fs, formatP, args)
```

Parameters

→ *fs*
An output stream to which to write the formatted output. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

→ *formatP*
A pointer to a format string that controls how subsequent arguments are converted for output.

→ *args*
A pointer to a list of zero or more parameters to be formatted as specified by the `formatP` string.

Returns Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

Comments This function internally calls `StrVPrintF()` to do the formatting. See that function for details on which format specifications are supported.

vsprintf Macro

Purpose Calls `StrVPrintF()` to write formatted output to the stdout output stream.

Declared In `StdIOPalm.h`

Prototype

```
#define vsprintf (fs, formatP, args)
```

Parameters

→ *fs*
An output stream to which to write the formatted output. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

→ *formatP*
A pointer to a format string that controls how subsequent arguments are converted for output.

→ *args*
A pointer to a list of zero or more parameters to be formatted as specified by the `y` string.

Returns Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

Comments See `StrVPrintF()` for details on which format specifications are supported.

Standard IO Provider Functions

These functions are used by a standard IO provider application.

SioClearScreen Function

Purpose Clears the entire standard IO output field.

Declared In `StdIOProvider.h`

Prototype

```
void SioClearScreen (void)
```

Parameters `None.`

Returns Returns nothing.

SioExecCommand Function

Purpose Executes a command line.

Declared In `StdIOProvider.h`

Prototype

```
Int16 SioExecCommand (const Char *cmd)
```

Parameters → *cmd*

A pointer to a string containing the command line to execute.

Returns Returns a value >= 0 on success or < 0 on failure.

Comments This function first looks for a built-in command with the specified name. If none is found, it looks for a Stdio application database with the name "Cmd-*cmdname*" where *cmdname* is the first word in the command string `cmd`.

If you pass the string "help" or "?" for the `cmd` parameter, `SioExecCommand` causes a help string to be printed for each built-in command. It actually executes each built-in command, passing the string "?" as `argv[1]`. Each command should handle this argument by printing a help line.

The `SioExecCommand` function is faster than calling `system()` to execute a command. However, `SioExecCommand` can be called only by the standard IO provider application, not the standard IO application.

SioFree Function

Purpose Closes down the standard IO manager.

Declared In `StdIOProvider.h`

Prototype

```
Err SioFree (void)
```

Parameters `None.`

Returns Returns 0 on success.

SioHandleEvent Function

Purpose Handles an event in the form that contains the standard IO output field and scroll arrows if the event belongs to the text field or scroll arrows.

Declared In `StdIOProvider.h`

Prototype
```
Boolean SioHandleEvent (SysEventType *event)
```

Parameters

→ *event*
Pointer to an `EventType` structure.

Returns Returns `true` if the event was handled and should not be processed by the application's own form event handler; returns `false` otherwise.

Comments This function must be called from the form event handler before it does its own processing with any of the objects unrelated to standard IO in the form.

SioInit Function

Purpose Initializes the standard IO manager.

Declared In `StdIOProvider.h`

Prototype
```
Err SioInit (UInt16 formID, UInt16 fieldID,
   UInt16 scrollerID)
```

Parameters

→ *formID*
The ID of the form that contains the input/output field.

→ *fieldID*
The ID of the field to be used for input/output.

→ *scrollerID*
The ID of the scroller associated with the input/output form.

Returns Returns 0 on success.

Application-Defined Function

You must supply this function in your stdio application.

SioMain Function

Purpose The main entry point for the stdio application.

Declared In `StdIOPalm.h`

Prototype

```
Int16 SioMain (UInt16 argc, const Char *argv[])
```

Parameters

→ *argc*
The number of parameters passed on the command line.

→ *argv*
An array of character pointers, one for each parameter passed on the command line.

Returns The return value from this routine is passed back to the system call that invoked it. Return 0 for no error.

52

String Manager

This chapter provides reference material for the string manager. The string manager API is declared in the header file `StringMgr.h`.

For more information, see Chapter 8, "Text," on page 259 of the *Palm OS Programmer's Companion*, vol. I.

String Manager Functions

StrAToI Function

Purpose Convert a string to an integer.

Declared In `StringMgr.h`

Prototype

```
Int32 StrAToI (const Char *str)
```

Parameters

→ *str*
Pointer to a string to convert.

Returns Returns the integer.

Comments Use this function instead of the standard `atoi` routine.

StrCaselessCompare Function

Purpose Compare two strings with case and accent insensitivity.

Declared In `StringMgr.h`

Prototype

```
Int16 StrCaselessCompare (const Char *s1,
    const Char *s2)
```

Parameters

→ *s1*
Pointer to a string.

→ *s2*
Pointer to a string.

Returns Returns 0 if the strings match.

Returns a positive number if `s1` > `s2`.

Returns a negative number if `s1` < `s2`.

Comments Use this function instead of the standard `stricmp` routine.

To support systems that use multi-byte character encodings, consider using `TxtCaselessCompare()` instead of this function (or `TxtCompare()` for a case-sensitive comparison). Both functions can match single-byte characters with their multi-byte equivalents, but `TxtCaselessCompare` can also return the length of the matching text.

See Also `StrNCaselessCompare()`, `TxtCaselessCompare()`, `StrCompare()`, `StrNCompare()`

StrCat Function

Purpose Concatenate one null-terminated string to another.

Declared In `StringMgr.h`

Prototype

```
Char *StrCat (Char *dst, const Char *src)
```

Parameters

→ *dst*
Pointer to the null-terminated destination string.

→ *src*
Pointer to the null-terminated source string.

Returns Returns a pointer to the destination string.

Comments Use this function instead of the standard `strcat` routine.

StrChr Function

Purpose Look for a character within a string.

Declared In `StringMgr.h`

Prototype

```
Char *StrChr (const Char *str, WChar chr)
```

Parameters

→ *str*
Pointer to the string to be searched.

→ *chr*
Character to search for.

Returns Returns a pointer to the first occurrence of character in `str`. Returns `NULL` if the character is not found.

Comments Use this function instead of the standard `strchr` routine.

This function can handle both single-byte characters and multi-byte characters correctly. However, you should make sure that you pass a `WChar` variable to `StrChr` instead of a `Char`. If you pass a `Char` variable, the function sign-extends the variable to a `WChar`, which causes problems if the value is 0x80 or higher.

Compatibility This routine does not correctly find a '\0' character on Palm OS® version 1.0.

See Also `StrStr()`

StrCompare Function

Purpose Compare two strings.

Declared In `StringMgr.h`

Prototype

```
Int16 StrCompare (const Char *s1, const Char *s2)
```

Parameters

→ *s1*
Pointer to a string.

→ *s2*
Pointer to a string.

Returns Returns 0 if the strings match.

Returns a positive number if `s1` sorts after `s2` alphabetically.

Returns a negative number if `s1` sorts before `s2` alphabetically.

Comments Use this function or `StrCompareAscii()` instead of the standard `strcmp` routine. This function is case sensitive.

To support systems that use multi-byte character encodings, consider using `TxtCompare()` instead of this function. Both functions can match single-byte characters with their multi-byte equivalents, but `TxtCompare` can also return the length of the matching text.

Compatibility Prior to Palm OS 4.0, `StrCompare` and `TxtCompare()` only performed one level of comparison and returned as soon as they found two unequal characters. For example, if you compared the string "celery" with the string "Cauliflower," both functions returned a value indicating that "celery" should appear before "Cauliflower" because they sorted "c" before "C."

In Palm OS 4.0, `StrCompare` calls `TxtCompare`, and `TxtCompare` performs a comparison using up to six comparison tables for sorting with increasing precision. As a result, in Palm OS 4.0 and higher, `StrCompare` sorts "Cauliflower" before "celery."

See Also `StrNCompare()`, `StrNCaselessCompare()`, `TxtCaselessCompare()`

StrCompareAscii Function

Purpose Compare two ASCII strings.

Declared In `StringMgr.h`

Prototype

```
Int16 StrCompareAscii (const Char *s1,
   const Char *s2)
```

Parameters

→ *s1*
Pointer to a string.

→ *s2*
Pointer to a string.

Returns Returns 0 if the strings match.

Returns a positive number if `s1` sorts after `s2` alphabetically.

Returns a negative number if `s1` sorts before `s2` alphabetically.

Comments Use this function instead of the standard `strcmp` routine. Use it to do case-sensitive comparisons on strings that are guaranteed to be 7-bit ASCII strings. This function performs a fast, simple byte-to-byte comparison that is guaranteed never to change.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `StrCompare()`, `StrNCompare()`, `TxtCompare()`, `StrCaselessCompare()`, `StrNCaselessCompare()`, `TxtCaselessCompare()`, `StrNCompareAscii()`

StrCopy Function

Purpose Copy one string to another.

Declared In `StringMgr.h`

Prototype

```
Char *StrCopy (Char *dst, const Char *src)
```

Parameters

→ *dst*
Pointer to the destination string.

→ *src*
Pointer to the source string.

Returns Returns a pointer to the destination string.

Comments Use this function instead of the standard `strcpy` routine.

This function does not work properly with overlapping strings.

StrDelocalizeNumber Function

Purpose Delocalize a number passed in as a string. Convert the number from any localized notation to US notation (decimal point and thousandth comma). The current thousand and decimal separators have to be passed in.

Declared In `StringMgr.h`

Prototype

```
Char *StrDelocalizeNumber (Char *s,
   Char thousandSeparator, Char decimalSeparator)
```

Parameters

↔ *s*
Pointer to the number as an ASCII string.

→ *thousandSeparator*
Current thousand separator.

→ *decimalSeparator*
Current decimal separator.

Returns Returns a pointer to the changed number and modifies the string in `s`.

Comments The current `thousandSeparator` and `decimalSeparator` can be determined by obtaining the value of the `prefNumberFormat` preference using `PrefGetPreference()` and then passing the

returned NumberFormatType to LocGetNumberSeparators(). For example:

```
Char *localizedNum;
NumberFormatType numFormat;
Char thousandsSeparator, decimalSeparator;

numFormat = (NumberFormatType)
  PrefGetPreference(prefNumberFormat);
LocGetNumberSeparators(numFormat,
  &thousandsSeparator, &decimalSeparator);
StrDelocalizeNumber(localizedNum,
  thousandsSeparator, decimalSeparator);
```

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also StrLocalizeNumber(), LocGetNumberSeparators()

StrIToA Function

Purpose Convert an integer to ASCII.

Declared In `StringMgr.h`

Prototype

```
Char *StrIToA (Char *s, Int32 i)
```

Parameters

← *s*

Pointer to a string of size `maxStrIToALen` in which to store the results.

→ *i*

Integer to convert.

Returns Returns a pointer to the result string.

See Also StrAToI(), StrIToH()

StrIToH Function

Purpose Convert an integer to hexadecimal ASCII.

Declared In `StringMgr.h`

Prototype
```
Char *StrIToH (Char *s, UInt32 i)
```

Parameters

← *s*
Pointer to a string in which to store the results.

→ *i*
Integer to convert.

Returns Returns the string pointer `s`.

See Also `StrIToA()`

StrLen Function

Purpose Compute the length of a string.

Declared In `StringMgr.h`

Prototype
```
UInt16 StrLen (const Char *src)
```

Parameters

→ *src*
Pointer to a string.

Returns Returns the length of the string in bytes.

Comments Use this function instead of the standard `strlen` routine.

This function returns the length of the string in bytes. On systems that support multi-byte characters, the number returned does not always equal the number of characters.

Compatibility In Palm OS 3.5 and Palm OS 4.x this function was declared to return an `Int16`. In Palm OS Garnet, and prior to Palm OS 3.5, this function returns a `UInt16`.

StrLocalizeNumber Function

Purpose Convert a number (passed in as a string) to localized format, using a specified thousands separator and decimal separator.

Declared In `StringMgr.h`

Prototype

```
Char *StrLocalizeNumber (Char *s,
    Char thousandSeparator, Char decimalSeparator)
```

Parameters

↔ *s*
Numeric ASCII string to localize.

→ *thousandSeparator*
Localized thousand separator.

→ *decimalSeparator*
Localized decimal separator.

Returns Returns a pointer to the changed number. Converts the number string in `s` by replacing all occurrences of "," with `thousandSeparator` and all occurrences of "." with `decimalSeparator`.

Comments The current `thousandSeparator` and `decimalSeparator` can be determined by obtaining the value of the `prefNumberFormat` preference using `PrefGetPreference()` and then passing the returned `NumberFormatType` to `LocGetNumberSeparators()`. For example:

```
Char *localizedNum;
NumberFormatType numFormat;
Char thousandsSeparator, decimalSeparator;

numFormat = (NumberFormatType)
  PrefGetPreference(prefNumberFormat);
LocGetNumberSeparators(numFormat,
  &thousandsSeparator, &decimalSeparator);
StrLocalizeNumber(localizedNum,
  thousandsSeparator, decimalSeparator);
```

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `StrDelocalizeNumber()`

StrNCaselessCompare Function

Purpose Compares two strings out to *n* characters with case and accent insensitivity.

Declared In `StringMgr.h`

Prototype

```
Int16 StrNCaselessCompare (const Char *s1,
    const Char *s2, Int32 n)
```

Parameters

→ *s1*
Pointer to the first string.

→ *s2*
Pointer to the second string.

→ *n*
Length in bytes of the text to compare.

Returns Returns 0 if the strings match.

Returns a positive number if `s1` > `s2`.

Returns a negative number if `s1` < `s2`.

Comments To support systems that use multi-byte character encodings, consider using `TxtCaselessCompare()` instead of this function (or `TxtCompare()` for a case-sensitive comparison). Both functions can match single-byte characters with their multi-byte equivalents, but `TxtCaselessCompare` can also return the length of the matching text.

Compatibility Implemented only if 2.0 New Feature Set is present. As of Palm OS 4.0, both `s1` and `s2` must be null-terminated strings.

See Also `StrNCompare()`, `StrCaselessCompare()`, `TxtCaselessCompare()`, `StrCompare()`

StrNCat Function

Purpose Concatenates one string to another clipping the destination string to a maximum of *n* bytes (including the null character at the end).

IMPORTANT: The Palm OS implementation of `StrNCat` differs from the implementation in the standard C library. See the Comments section for details.

Declared In `StringMgr.h`

Prototype

```
Char *StrNCat (Char *dst, const Char *src,
    Int16 n)
```

Parameters

→ *dst*
Pointer to the null-terminated destination string.

→ *src*
Pointer to the source string.

→ *n*
Maximum length in bytes for `dst`, including the terminating null character.

Returns Returns a pointer to the destination string.

Comments This function differs from the standard C `strncat` function in these ways:

- `StrNCat` treats the parameter `n` as the maximum length in bytes for `dst`. That means it will copy at most `n - StrLen(dst) - 1` bytes from `src`. The standard C function always copies `n` bytes from `src` into `dst`. (It copies the entire `src` into `dst` if the length of `src` is less than `n`).
- If the length of the destination string reaches `n - 1`, `StrNCat` stops copying bytes from `src` and appends the terminating null character to `dst`. If the length of the destination string is already greater than or equal to `n - 1` before the copying begins, `StrNCat` does not copy any data from `src`.
- In the standard C function, if `src` is less than `n`, the entire `src` string is copied into `dst` and then the remaining space is filled with null characters. `StrNCat` does not fill the remaining space with null characters in released ROMs. In debug ROMs, `StrNCat` fills the remaining bytes with the value `0xFE`.

On systems with multi-byte character encodings, this function makes sure that it does not copy part of a multi-byte character. If the last byte copied from `src` contains the high-order or middle byte of a multi-byte character, `StrNCat` backs up in `dst` until the byte after the end of the previous character, and replaces that byte with a null character.

Compatibility Implemented only if 2.0 New Feature Set is present.

StrNCompare Function

Purpose Compare two strings out to *n* bytes. This function is case and accent sensitive.

Declared In `StringMgr.h`

Prototype

```
Int16 StrNCompare (const Char *s1,
   const Char *s2,
   Int32 n)
```

Parameters

→ *s1*
Pointer to a string.

→ *s2*
Pointer to a string.

→ *n*
Length in bytes of text to compare.

Returns Returns 0 if the strings match.

Returns a positive number if `s1` > `s2`.

Returns a negative number if `s1` < `s2`.

Comments To support systems that use multi-byte character encodings, consider using `TxtCompare()` instead of this function. Both functions can match single-byte characters with their multi-byte equivalents, but `TxtCompare` can also return the length of the matching text.

Compatibility Implemented only if 2.0 New Feature Set is present. As of Palm OS 4.0, both `s1` and `s2` must be null-terminated strings.

See Also `StrCompare()`, `StrNCaselessCompare()`, `StrCaselessCompare()`, `TxtCaselessCompare()`, `StrNCompareAscii()`

StrNCompareAscii Function

Purpose Compare two ASCII strings out to *n* bytes.

Declared In `StringMgr.h`

Prototype

```
Int16 StrNCompareAscii (const Char *s1,
    const Char *s2, Int32 n)
```

Parameters

→ *s1*
Pointer to a string.

→ *s2*
Pointer to a string.

→ *n*
Length in bytes of text to compare.

Returns Returns 0 if the strings match.

Returns a positive number if `s1` sorts after `s2` alphabetically.

Returns a negative number if `s1` sorts before `s2` alphabetically.

Comments Use this function instead of the standard `strncmp` routine. Use it to do case-sensitive comparisons on strings that are guaranteed to be 7-bit ASCII strings. This function performs a fast, simple byte-to-byte comparison that is guaranteed never to change.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `StrCompare()`, `StrNCompare()`, `TxtCompare()`, `StrCaselessCompare()`, `StrNCaselessCompare()`, `TxtCaselessCompare()`, `StrCompareAscii()`

StrNCopy Function

Purpose Copies up to *n* bytes from a source string to the destination string. Terminates `dst` string at index *n*-1 if the source string length was *n*-1 or less.

Declared In `StringMgr.h`

Prototype

```
Char *StrNCopy (Char *dst, const Char *src,
    Int16 n)
```

Parameters

→ *dst*
Pointer to the destination string.

→ *src*
Pointer to the source string.

→ *n*
Maximum number of bytes to copy from `src` string.

Returns Returns nothing.

Comments On systems with multi-byte character encodings, this function makes sure that it does not copy part of a multi-byte character. If the nth byte of `src` contains the high-order or middle byte of a multi-byte character, `StrNCopy` backs up in `dst` until the byte after the end of the previous character, and replaces the remaining bytes (up to n-1) with nulls.

Be aware that the nth byte of `dst` upon return may contain the last byte of a multi-byte character. If you plan to terminate the string by setting its last character to NULL, you must not pass the entire length of the string to `StrNCopy`. If you do, your code may overwrite the final byte of the last character.

```
// WRONG! You may overwrite part of multi-byte
// character.
Char dst[n];
StrNCopy(dst, src, n);
dst[n-1] = chrNull;
```

Instead, if you write to the last byte of the destination string, pass one less than the size of the string to `StrNCopy`.

```
// RIGHT. Instead pass n-1 to StrNCopy.
Char dst[n];
StrNCopy(dst, src, n-1);
dst[n-1] = chrNull;
```

Compatibility Implemented only if 2.0 New Feature Set is present.

StrPrintF Function

Purpose Implements a subset of the ANSI C `sprintf` call, which writes formatted output to a string.

Declared In `StringMgr.h`

Prototype

```
Int16 StrPrintF (Char *s,
   const Char *formatStr, ...)
```

Parameters

→ *s*
Pointer to a string into which the results are written.

→ *formatStr*
Pointer to the format specification string.

`...`
Zero or more arguments to be formatted as specified by `formatStr`.

Returns Number of characters written to destination string. Returns a negative number if there is an error.

Comments This function internally calls `StrVPrintF()` to do the formatting. See that function's description—specifically, Table 52.1 on page 1011—for details on which format specifications are supported.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `StrVPrintF()`

StrStr Function

Purpose Look for a substring within a string.

Declared In `StringMgr.h`

Prototype

```
Char *StrStr (const Char *str, const Char *token)
```

Parameters

→ *str*
Pointer to the string to be searched.

→ *token*
Pointer to the string to search for.

Returns Returns a pointer to the first occurrence of `token` in `str` or `NULL` if not found.

Comments Use this function instead of the standard `strstr` routine.

On systems with multi-byte character encodings, this function makes sure that it does not match only part of a multi-byte character. If the matching strings begins at an inter-character boundary, then this function returns `NULL`.

NOTE: If the value of the `token` parameter is the empty string, this function returns `NULL`. This is different than the standard `strstr` function, which returns `str` when `token` is the empty string.

See Also `StrChr()`

StrToLower Function

Purpose Convert all the characters in a string to lowercase.

Declared In `StringMgr.h`

Prototype

```
Char *StrToLower (Char *dst, const Char *src)
```

Parameters

→ *dst*
Pointer to a string.

→ *src*
Pointer to a null-terminated string.

Returns Returns a pointer to the destination string.

Compatibility Prior to Palm OS version 3.5, this function only converted accented characters on Japanese devices. On Palm OS version 3.5 and higher, all characters are appropriately lowercased, including accented characters on Latin devices.

StrVPrintF Function

Purpose Implements a subset of the ANSI C `vsprintf` call, which writes formatted output to a string.

Declared In `StringMgr.h`

Prototype

```
Int16 StrVPrintF (Char *s, const Char *formatStr,
    va_list arg)
```

Parameters

← *s*
: Pointer to a string into which the results are written. This string is always terminated by a null terminator.

→ *formatStr*
: Pointer to the format specification string.

→ *arg*
: Pointer to a list of zero or more parameters to be formatted as specified by the `formatStr` string.

Returns Number of characters written to destination string, not including the null terminator. Returns a negative number if there is an error.

Comments Like the C `vsprintf` function, this function is designed to be called by your own function that takes a variable number of arguments and passes them to this function. For details on how to use it, see "Using the StrVPrintF Function" on page 275 of the *Palm OS Programmer's Companion*, vol. I, or refer to `vsprintf` in a standard C reference book.

Currently, only the conversion specifications `%d`, `%i`, `%u`, `%x`, `%s`, and `%c` are implemented by `StrVPrintF` (and related functions). Optional modifiers that are supported include: `+`, `-`, <space>, `*`, <digits>, `h` and `l` (long). Following is a brief description of how these format specifications work (see a C book for more details).

Each conversion specification begins with the % character. Following the % character, there may be one or more of the characters list in Table 52.1, in sequence.

Table 52.1 StrVPrintF Format Specification

Character	Description

+	Specifies that a sign always be placed before a number produced by a signed conversion. A + overrides a space if both are used. Example:
	`StrPrintF(s,"%+d  %+d",4,-5);`
	Output to `s`:
	`+4  -5`

-	Specifies that the printed value is left justified within the field width allowed for it. Example:
	`StrPrintF(s,"%5d%-5d%d",6,9,8);`
	Output to `s`:
	`    69    8`

<space>	Specifies that a minus sign always be placed before a negative number and a space before a positive number. Example:
	`StrPrintF(s,"% d  % d",4,-5);`
	Output to `s`:
	`  4  -5`

*	Indicates that the next argument (must be an integer) in the list specifies the field width. In this case, the argument following that one is used for the value of this field. Example:
	`StrPrintF(s,"%*d%d",4,8,5);`
	Output to `s`:
	`8    5`

Table 52.1 StrVPrintF Format Specification *(continued)*

Character	Description

<number>	Specifies a minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces on the left (or right, if the left justified flag is also specified) to fill out the field width. Example:
	`StrPrintF(s,"%d%5d",4,3);`
	Output to `s`:
	`4  3`

`h`	Specifies that the following d, i, u, or x conversion corresponds to a `short` or `unsigned short` argument. Example:
	`StrPrintF(s,"%hd",401);`
	Output to `s`:
	`401`

`l` or `L`	Specifies that the following d, i, u, x, or c conversion corresponds to a `long` or `unsigned long`
	`StrPrintF(s,"%ld",999999999);`
	Output to `s`:
	`999999999`

<character>	A character that indicates the type of conversion to be performed. The supported conversion characters include:

`d` or `i`	A signed integer argument is converted to decimal notation. Example:
	`StrPrintF(s,"%d  %d",4,-4);`
	Output to `s`:
	`4  -4`

Table 52.1 StrVPrintF Format Specification *(continued)*

Character		Description

	`u`	An unsigned integer argument is converted to decimal notation. Example:
		`StrPrintF(s,"%u  %u",4,-4);`
		Output to `s`:
		`4  65532`

	`x`	An integer argument is converted to hexadecimal notation. Example:
		`StrPrintF(s,"%x",125);`
		Output to `s`:
		`0000007D`

	`s`	A string (`char *`) argument is copied to the destination string. Example:
		`StrPrintF(s,"ABC%s","DEF");`
		Output to `s`:
		`ABCDEF`

	`c` or `C`	A single character argument is copied to the destination string. If `C` is used or if the `l` modifier is used, the argument must be a `WChar`. Example:
		`StrPrintF(s,"Telephone%c",'s');`
		Output to `s`:
		`Telephones`

	`%`	A % character is copied to the destination string. Example:
		`StrPrintF(s,"%%");`
		Output to `s`:
		`%`

Example Here's an example of how to use this call:

```
#include <unix_stdarg.h>
void MyPrintF(Char *s, Char *formatStr, ...)
{
  va_list args;
  Char text[0x100];
  va_start(args, formatStr);
  StrVPrintF(text, formatStr, args);
  va_end(args);
  MyPutS(text);
}
```

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `StrPrintF()`

53

System Event Manager

This chapter describes functions available in the system event manager. The system event manager API is declared in the header files `Event.h` and `SysEvtMgr.h`.

For more information on the system event manager, see the chapter "Event Loop" in the *Palm OS Programmer's Companion*, vol. I. The reference for specific events sent by the system are documented in "Palm OS Events."

System Event Manager Data Structures

The following system event manager data structures are documented in the "Palm OS Events" chapter:

- `eventsEnum`
- `EventType`
- `EventPtr`

System Event Manager Functions

EvtAddEventToQueue Function

Purpose Add an event to the event queue.

Declared In `Event.h`

Prototype

```
void EvtAddEventToQueue (const EventType *event)
```

Parameters → *event*
Pointer to the structure that contains the event.

Returns Returns nothing.

Comments This function makes a copy of the structure that you pass in and adds it to the event queue.

EvtAddUniqueEventToQueue Function

Purpose Add an event to the event queue, replacing one of the same type if it is found.

Declared In `Event.h`

Prototype

```
void EvtAddUniqueEventToQueue
    (const EventType *eventP, UInt32 id,
    Boolean inPlace)
```

Parameters

→ *eventP*
Pointer to the structure that contains the event.

→ *id*
ID of the event. 0 means match only on the type.

→ *inPlace*
If `true`, any existing event is replaced. If `false`, the existing event is deleted and a new event is added to end of queue.

Returns Returns nothing.

Comments This function looks for an event in the event queue of the same event type and ID (if specified). The routine replaces it with the new event, if found.

If no existing event is found, the new event is copied to the queue.

If an existing event is found, the routine proceeds as follows:

- If `inPlace` is `true`, the existing event is replaced with a copy of the new event.
- If `inPlace` is `false`, the existing event is removed and the new event is added to the end of the queue.

Compatibility Implemented only if 2.0 New Feature Set is present.

EvtCopyEvent Function

Purpose Copy an event.

Declared In `Event.h`

Prototype

```
void EvtCopyEvent (const EventType *source,
   EventType *dest)
```

Parameters

→ *source*
Pointer to the structure containing the event to copy.

← *dest*
Pointer to the structure to copy the event to.

Returns Returns nothing.

Comments Use this function only if you want to create an event that has the same type as the `source` event. The `data` field in an `EventType` structure is specific to events of a given type. If you were to use this function to copy a `keyDownEvent` and then change it to a `frmLoadEvent`, the resulting `frmLoadEvent` would not have the proper `data` field.

If you want to create an event of a different type, do not use `EvtCopyEvent`. First clear the `EventType` structure using `MemSet()` and then change the event type:

```
MemSet(&event, sizeof(EventType), 0);
event.eType = frmLoadEvent;
event.data.frmLoad.formID = formID;
EvtAddEventToQueue(&event);
```

EvtDequeuePenPoint Function

Purpose Get the next pen point out of the pen queue. This function is called by recognizers.

Declared In `SysEvtMgr.h`

Prototype

```
Err EvtDequeuePenPoint (PointType *retP)
```

Parameters

← *retP*
Return point.

Returns Always returns 0.

Comments Called by a recognizer that wishes to extract the points of a stroke. Returns the point (-1, -1) at the end of a stroke.

Before calling this routine, you must call `EvtDequeuePenStrokeInfo()`.

EvtDequeuePenStrokeInfo Function

Purpose Initiate the extraction of a stroke from the pen queue.

Declared In `SysEvtMgr.h`

Prototype

```
Err EvtDequeuePenStrokeInfo (PointType *startPtP,
    PointType *endPtP)
```

Parameters

← *startPtP*
Start point returned here.

← *endPtP*
End point returned here.

Returns Always returns 0.

Comments Called by the system function `EvtGetSysEvent` when a `penUpEvent` is being generated. This routine must be called before `EvtDequeuePenPoint()` is called.

Subsequent calls to `EvtDequeuePenPoint` return points at the starting point in the stroke and including the end point. After the end point is returned, the next call to `EvtDequeuePenPoint()` returns the point -1, -1.

See Also `EvtDequeuePenPoint()`

EvtEnableGraffiti Function

Purpose Enable or disable Graffiti® or Graffiti 2 handwriting recognition.

Declared In `SysEvtMgr.h`

Prototype

```
void EvtEnableGraffiti (Boolean enable)
```

Parameters

→ *enable*
`true` to enable handwriting recognition, `false` to disable it.

Returns Returns nothing.

EvtEnqueueKey Function

Purpose Place keys into the key queue.

Declared In `SysEvtMgr.h`

Prototype

```
Err EvtEnqueueKey (WChar ascii, UInt16 keycode,
    UInt16 modifiers)
```

Parameters

→ *ascii*
Character code for the key.

→ *keycode*
Virtual key code of key. This is the `keyCode` field of the `keyDownEvent` and is currently unused.

→ *modifiers*
Modifiers for `keyDownEvent`.

Returns Returns 0 if successful, or `evtErrParamErr` if an error occurs.

Comments This function disables interrupts while the queue header is being modified because both interrupt- and non-interrupt-level code can post keys into the queue.

Entries in the key queue only take 1 byte if the `ascii` parameter has a value less than 256 and the `keycode` and `modifiers` parameters are both zero. Otherwise an entry can take up to 7 bytes.

IMPORTANT: Make sure you pass a `WChar` as the `ascii` parameter, not a `Char`. If you pass a high-ASCII `Char`, the compiler sign-extends it to be a 16-bit value, resulting in the wrong character being added to the key queue.

EvtEventAvail Function

Purpose Return `true` if an event is available.

Declared In `Event.h`

Prototype

```
Boolean EvtEventAvail (void)
```

Parameters `None.`

Returns Returns `true` if an event is available, `false` otherwise.

Compatibility Implemented only if 2.0 New Feature Set is present.

EvtFlushKeyQueue Function

Purpose Flush all keys out of the key queue.

Declared In `SysEvtMgr.h`

Prototype
```
Err EvtFlushKeyQueue (void)
```

Parameters `None.`

Returns Always returns 0.

Comments Called by the system function `EvtSetPenQueuePtr`.

EvtFlushNextPenStroke Function

Purpose Flush the next stroke out of the pen queue.

Declared In `SysEvtMgr.h`

Prototype
```
Err EvtFlushNextPenStroke ()
```

Parameters `None.`

Returns Always returns 0.

Comments Called by recognizers that need only the start and end points of a stroke. If a stroke has already been partially dequeued (by `EvtDequeuePenStrokeInfo()`) this routine finishes the stroke dequeueing. Otherwise, this routine flushes the next stroke in the queue.

See Also `EvtDequeuePenPoint()`

EvtFlushPenQueue Function

Purpose Flush all points out of the pen queue.

Declared In `SysEvtMgr.h`

Prototype
```
Err EvtFlushPenQueue (void)
```

Parameters `None`

Returns Always returns 0.

Comments Called by the system function `EvtSetKeyQueuePtr`.

See Also `EvtPenQueueSize()`

EvtGetEvent Function

Purpose Return the next available event.

Declared In `Event.h`

Prototype

```
void EvtGetEvent (EventType *event, Int32 timeout)
```

Parameters

← *event*
Pointer to the structure to hold the event returned.

→ *timeout*
Maximum number of ticks to wait before an event is returned (`evtWaitForever` means wait indefinitely).

Comments Pass `evtWaitForever` as the timeout in most instances. When running on the device, this makes the CPU go into doze mode until the user provides input. For applications that do animation, pass a `timeout` value greater than or equal to zero.

Note that a timeout value greater than or equal to zero is simply the *maximum* number of ticks which can elapse before `EvtGetEvent` returns an event. If any other event—including a `nilEvent`—occurs before this time has elapsed, `EvtGetEvent` will return that event. Otherwise, once the specified time has elapsed `EvtGetEvent` generates and returns a `nilEvent`. If you supply a value of zero for the timeout parameter, `EvtGetEvent` returns the event currently in the queue, or, if there aren't any events in the queue, it immediately generates and returns a `nilEvent`.

Returns Returns nothing.

EvtGetPen Function

Purpose Return the current status of the pen.

Declared In `Event.h`

Prototype

```
void EvtGetPen (Int16 *pScreenX, Int16 *pScreenY,
    Boolean *pPenDown)
```

Parameters

← *pScreenX*
x location relative to display.

← *pScreenY*
y location relative to display.

← *pPenDown*
`true` or `false`.

Returns Returns nothing.

Comments Called by various UI routines.

See Also `EvtGetPenNative()`, `KeyCurrentState()`

EvtGetPenBtnList Function

Purpose Return a pointer to the silk-screen button array.

Declared In `SysEvtMgr.h`

Prototype

```
const PenBtnInfoType *EvtGetPenBtnList
    (UInt16 *numButtons)
```

Parameters ← *numButtons*
The number of elements in the returned array.

Returns Returns a pointer to an array of the silk-screen buttons.

Comments This function returns an array of `PenBtnInfoType` structures:

Prototype

```
typedef struct PenBtnInfoType {
  RectangleType boundsR;
  WChar asciiCode;
  UInt16 keyCode;
  UInt16 modifiers;
} PenBtnInfoType;
```

Fields

`boundsR`
The button's bounding rectangle.

`asciiCode`
The character code generated when the button is tapped. This is typically a virtual character.

`keyCode`
Currently unused.

`modifiers`
Modifiers for the key down event. (See the description of the modifiers field for `keyDownEvent`.)

The number of buttons is device-dependent. Even devices with the same Palm OS® version may have differing numbers of silk-screen buttons. For example, Japanese devices typically have four extra

silk-screen buttons used to transliterate characters into different alphabets.

Compatibility If Palm OS Cobalt Feature Set is present this function is unimplemented.

See Also `EvtProcessSoftKeyStroke()`

EvtGetPenNative Function

Purpose Get the current status of the pen using a window's active coordinate system.

Declared In `Window.h`

Prototype

```
void EvtGetPenNative (WinHandle winH,
    Int16 *pScreenX, Int16 *pScreenY,
    Boolean *pPenDown)
```

Parameters

→ *winH*
Handle to a valid window.

← *pScreenX*
x location relative to the window.

← *pScreenY*
y location relative to the window.

← *pPenDown*
`true` if the pen is down, `false` otherwise.

Returns Returns nothing.

Comments This function is a variation on `EvtGetPen()`. `EvtGetPen` returns a pen sample using the standard coordinate system, relative to the draw window, whereas `EvtGetPenNative` returns a pen sample using the active coordinate system of `winH`, relative to the window origin. If the active coordinate system is high density, the returned pen sample uses high-density coordinates.

On a debug ROM this function displays an error if `winH` doesn't reference a valid window object.

Compatibility Implemented only if the High-Density Display Feature Set is present.

EvtGetSilkscreenAreaList Function

Purpose Returns a pointer to the silk-screen area array. This array contains the bounds of each silk-screen area.

Declared In `SysEvtMgr.h`

Prototype

```
const SilkscreenAreaType *EvtGetSilkscreenAreaList
    (UInt16 *numAreas)
```

Parameters

← *numAreas*
The number of elements in the returned array.

Returns Returns a pointer to an array containing the bounds of each silk-screen area.

Comments This function returns an array of the `SilkscreenAreaType` structures:

Prototype

```
typedef struct SilkscreenAreaType {
  RectangleType bounds;
  UInt32 areaType;
  UInt16 index;
  } SilkscreenAreaType;
```

Fields

`bounds`
The area's bounds.

`areaType`
The area type, can be one of the following:

`silkscreenRectGraffiti`
The input area.

`silkscreenRectScreen`
The entire silkscreen area.

Depending on the handheld manufacturer, Palm Powered™ devices may have other area types.

`index`
If the area type is `silkscreenRectGraffiti`, this field is either `alphaGraffitiSilkscreenArea` to represent the portion where letters are entered or `numericGraffitiSilkscreenArea` to represent the portion where numbers are entered.

Compatibility Implemented only if 3.5 New Feature Set is present. If 5.0 New Feature Set is present, this function should be considered "System Use Only"; applications should do what they can to avoid using it.

EvtKeydownIsVirtual Macro

Purpose Indicates if `eventP` is a pointer to a virtual character key down event.

Declared In `Event.h`

Prototype

```
#define EvtKeydownIsVirtual (eventP)
```

Parameters → *eventP*
Pointer to an `EventType` structure.

Returns Returns `true` if the character is a letter in an alphabet or a numeric digit, `false` otherwise.

Comments The macro assumes that the caller has already determined the event is a `keyDownEvent`.

This macro is intended for use by the system. Applications should use `TxtGlueCharIsVirtual()`, contained in the PalmOSGlue Library.

Compatibility Implemented in the Palm OS 3.5 SDK.

See Also `TxtGlueCharIsVirtual()`

EvtKeyQueueEmpty Function

Purpose Return `true` if the key queue is currently empty.

Declared In `SysEvtMgr.h`

Prototype

```
Boolean EvtKeyQueueEmpty (void)
```

Parameters `None.`

Returns Returns `true` if the key queue is currently empty, otherwise returns `false`.

Comments Usually called by the key manager to determine if it should enqueue auto-repeat keys.

EvtKeyQueueSize Function

Purpose Return the size of the current key queue in bytes.

Declared In `SysEvtMgr.h`

Prototype
```
UInt32 EvtKeyQueueSize (void)
```

Parameters `None.`

Returns Returns size of queue in bytes.

Comments Called by applications that wish to see how large the current key queue is.

Compatibility If Palm OS Cobalt Feature Set is present this function is unimplemented.

EvtPenQueueSize Function

Purpose Return the size of the current pen queue in bytes.

Declared In `SysEvtMgr.h`

Prototype
```
UInt32 EvtPenQueueSize (void)
```

Parameters `None.`

Returns Returns size of queue in bytes.

Comments Call this function to see how large the current pen queue is.

Compatibility If Palm OS Cobalt Feature Set is present this function is unimplemented.

EvtProcessSoftKeyStroke Function

Purpose Translate a stroke in the system area of the digitizer and enqueue the appropriate key events in to the key queue.

Declared In `SysEvtMgr.h`

Prototype
```
Err EvtProcessSoftKeyStroke (PointType *startPtP,
    PointType *endPtP)
```

Parameters → *`startPtP`*
Start point of stroke.

→ *endPtP*
End point of stroke.

Returns Returns 0 if recognized, -1 if not recognized.

Compatibility If Palm OS Cobalt Feature Set is present this function is unimplemented.

See Also `EvtGetPenBtnList()`, `GrfProcessStroke()`

EvtResetAutoOffTimer Function

Purpose Reset the auto-off timer.

Declared In `SysEvtMgr.h`

Prototype

```
Err EvtResetAutoOffTimer (void)
```

Parameters `None.`

Returns Always returns 0.

Comments Called by the serial link manager; can be called periodically by other managers.

`EvtResetAutoOffTimer` resets the auto-off timer so that the device does not turn off until at least the default amount of idle time has passed. You can use this function to ensure that the device doesn't automatically power off during a long operation without user input (for example, when there is a lot of serial port activity).

If you need more control over the auto-off timer and the 3.5 New Feature Set is present, consider using `EvtSetAutoOffTimer()` instead of this function.

See Also `SysSetAutoOffTime()`

EvtSetAutoOffTimer Function

Purpose Set the auto-off timer.

Declared In `SysEvtMgr.h`

Prototype

```
Err EvtSetAutoOffTimer (EvtSetAutoOffCmd cmd,
    UInt16 timeout)
```

Parameters

→ *cmd*
One of the defined commands.

→ *timeout*
A new timeout value in seconds. If `cmd` is `ResetTimer`, this parameter is ignored.

Returns Always returns `errNone`.

Comments Use `EvtSetAutoOffTimer` to ensure that the device doesn't automatically power off during a long operation that has no user input (for example, when there is a lot of serial port activity).

The `cmd` parameter specifies the operation you want to perform. It takes one of the following `EvtSetAutoOffCmd` constants:

`SetAtLeast`
Make sure that the device won't turn off until `timeout` seconds of idle time has passed. (This operation only changes the current value if it's less than the value you specify.)

`SetExactly`
Set the timer to turn off in `timeout` seconds

`SetAtMost`
Make sure the device will turn before `timeout` seconds has passed. (This operation only changes the current value if it's greater than the value you specify.)

`SetDefault`
Change the default auto-off timeout to `timeout` seconds.

`ResetTimer`
Reset the auto-off timer so that the device does not turn off until at least the default seconds of idle time has passed.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `EvtResetAutoOffTimer()`, `SysSetAutoOffTime()`

EvtSetNullEventTick Function

Purpose Make sure a `nilEvent` occurs by the specified time, specified in ticks since the last reset.

Declared In `SysEvtMgr.h`

Prototype

```
Boolean EvtSetNullEventTick(UInt32 tick)
```

Parameters → *tick*

The time, in ticks, since the last reset by which a `nilEvent` is to be added to the queue.

Returns Returns `true` if timeout value changed, or `false` if it did not change.

Compatibility In versions prior to Palm OS 3.5, this function was implemented as a macro.

EvtSysEventAvail Function

Purpose Return `true` if a low-level system event (such as a pen or key event) is available.

Declared In `SysEvtMgr.h`

Prototype

```
Boolean EvtSysEventAvail (Boolean ignorePenUps)
```

Parameters `ignorePenUps`

If `true`, this routine ignores pen-up events when determining if there are any system events available.

Returns Returns `true` if a system event is available.

Comments Call `EvtEventAvail()` to determine whether high-level software events are available.

Compatibility Implemented only if 2.0 New Feature Set is present.

EvtWakeup Function

Purpose Force the event manager to wake up and send a `nilEvent` to the current application.

Declared In `SysEvtMgr.h`

Prototype
```
Err EvtWakeup (void)
```

Parameters `None.`

Returns Always returns 0.

Comments Called by interrupt routines, like the sound manager and alarm manager.

See Also `EvtWakeupWithoutNilEvent()`

EvtWakeupWithoutNilEvent Function

Purpose Force the event manager to wake up without sending a `nilEvent` to the current application.

Declared In `SysEvtMgr.h`

Prototype
```
Err EvtWakeupWithoutNilEvent()
```

Parameters `None.`

Returns Always returns 0.

Comments Called by interrupt routines.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `EvtWakeup()`

54

System Manager

This chapter provides reference material for the System Manager. The System Manager API is declared in the header files `SystemMgr.h` and `SysUtils.h`.

For more information on the System Manager, see the chapters "Application Startup and Stop" and "Palm System Support" in the *Palm OS Programmer's Companion*, vol. I.

System Manager Data Structures

SysDBListItemType Struct

Purpose The `SysDBListItemType` structure describes a single database or panel. The `SysCreateDataBaseList()` and `SysCreatePanelList()` functions each create and return an array of `SysDBListItemType` structures.

Prototype

```
typedef struct {
  Char name[dmDBNameLength];
  UInt32 creator;
  UInt32 type;
  UInt16 version;
  LocalID dbID;
  UInt16 cardNo;
  BitmapPtr iconP;
} SysDBListItemType;
```

System Functions

SysAppLaunch Function

Purpose Launch a specified application as a subroutine of the caller.

Declared In `SystemMgr.h`

Prototype

```
Err SysAppLaunch (UInt16 cardNo, LocalID dbID,
   UInt16 launchFlags, UInt16 cmd, MemPtr cmdPBP,
   UInt32 *resultP)
```

Parameters

→ *cardNo, dbID*
The card number and ID of the resource database of the application to launch.

→ *launchFlags*
Set to 0.

→ *cmd*
Launch code.

→ *cmdPBP*
Launch code parameter block.

← *resultP*
The value returned from the application's `PilotMain()` routine.

Returns Returns 0 if no error, or one of `sysErrParamErr`, `memErrNotEnoughSpace`, or `sysErrOutOfOwnerIDs`.

Comments Applications can use `SysAppLaunch` to send a specific launch code to another application and have control return to the calling application when finished. This function in effect makes the specified application a subroutine of the caller. If you want to actually close your application and call another application, use `SysUIAppSwitch()` instead of this function. `SysUIAppSwitch` sends the current application an `appStopEvent` and then starts the specified application.

Do not use this function to open the system-supplied Application Launcher application. If another application has replaced the default launcher with one of its own, this function will open the custom launcher instead of the system-supplied one. To open the

system-supplied launcher reliably, enqueue a `keyDownEvent` that contains a `launchChr`, as shown in the section "Application Launcher" of the user interface chapter in the *Palm OS Programmer's Companion*, vol. I.

NOTE: For important information regarding the correct use of this function, see the "Application Startup and Stop" chapter in the *Palm OS Programmer's Companion*, vol. I.

See Also `SysBroadcastActionCode()`, `SysUIAppSwitch()`, `SysCurAppDatabase()`

SysBatteryInfo Function

Purpose Retrieve settings for the batteries. Set `set` to `false` to retrieve battery settings. (Applications should *not* change any of the settings).

WARNING! Use this function only to *retrieve* settings!

Declared In `SystemMgr.h`

Prototype

```
UInt16 SysBatteryInfo (Boolean set,
    UInt16 *warnThresholdP,
    UInt16 *criticalThresholdP, Int16 *maxTicksP,
    SysBatteryKind *kindP, Boolean *pluggedIn,
    UInt8 *percentP)
```

Parameters

`set`
If `false`, parameters with non-`NULL` pointers are retrieved. Never set this parameter to `true`.

`warnThresholdP`
Pointer to battery voltage warning threshold in volts*100, or `NULL`.

`criticalThresholdP`
Pointer to the battery voltage critical threshold in volts*100, or `NULL`.

`maxTicksP`
Pointer to the battery timeout, or `NULL`.

`kindP`
: Pointer to the battery kind, or `NULL`.

`pluggedIn`
: Pointer to `pluggedIn` return value, or `NULL`.

`percentP`
: Percentage of power remaining in the battery.

Returns Returns the current battery voltage in volts*100.

Comments Call this function to make sure an upcoming activity won't be interrupted by a low battery warning.

`warnThresholdP` and `maxTicksP` are the battery-warning voltage threshold and time out. If the battery voltage falls below the threshold, or the timeout expires, a `lowBatteryChr` key event is put on the queue. Normally, applications call `SysHandleEvent()` which calls `SysBatteryDialog` in response to this event.

`criticalThresholdP` is the battery voltage threshold. If battery voltage falls below this level, the system turns itself off without warning and doesn't turn on until battery voltage is above it again.

Compatibility This function was revised for Palm OS® 3.0. In Palm OS 3.0, the `percentP` parameter was added. This enhancement is implemented only if 3.0 New Feature Set is present.

See Also `SysBatteryInfoV20()`

SysBatteryInfoV20 Function

Purpose Retrieve settings for the batteries. Set to `false` to retrieve battery settings. (Applications should *not* change any of the settings).

WARNING! Use this function only to *retrieve* settings!

Declared In `SystemMgr.h`

Prototype

```
UInt16 SysBatteryInfoV20 (Boolean set,
   UInt16 *warnThresholdP,
   UInt16 *criticalThresholdP, Int16 *maxTicksP,
   SysBatteryKind *kindP, Boolean *pluggedIn)
```

Parameters

`set`
: If `false`, parameters with non-`NULL` pointers are retrieved. Never set this parameter to `true`.

`warnThresholdP`
: Pointer to battery voltage warning threshold in volts*100, or `NULL`.

`criticalThresholdP`
: Pointer to the battery voltage critical threshold in volts*100, or `NULL`.

`maxTicksP`
: Pointer to the battery timeout, or `NULL`.

`kindP`
: Pointer to the battery kind, or `NULL`.

`pluggedIn`
: Pointer to `pluggedIn` return value, or `NULL`.

Returns Returns the current battery voltage in volts*100.

Comments Call this function to make sure an upcoming activity won't be interrupted by a low battery warning.

`warnThresholdP` and `maxTicksP` are the battery-warning voltage threshold and time out. If the battery voltage falls below the threshold, or the timeout expires, a `lowBatteryChr` key event is put on the queue. Normally, applications call `SysHandleEvent()` which calls `SysBatteryDialog` in response to this event.

`criticalThresholdP` is the battery voltage threshold. If battery voltage falls below this level, the system turns itself off without warning and doesn't turn on until battery voltage is above it again.

Compatibility This function corresponds to the Palm OS 2.0 version of `SysBatteryInfo`. Implemented only if 3.0 New Feature Set is present.

See Also `SysBatteryInfo()`

SysBinarySearch Function

Purpose Search elements in a sorted array for the specified data according to the specified comparison function.

Declared In `SysUtils.h`

Prototype

```
Boolean SysBinarySearch (void const *baseP,
   UInt16 numOfElements, Int16 width,
   SearchFuncPtr searchF, void const *searchData,
   Int32 other, Int32 *position,
   Boolean findFirst)
```

Parameters

`baseP`
: Base pointer to an array of elements

`numOfElements`
: Number of elements to search. Must be greater than 0.

`width`
: Width of each array element.

`searchF`
: Search function.

`searchData`
: Data to search for. This data is passed to the `searchF` function.

`other`
: Data to be passed as the third parameter (the `other` parameter) to the comparison function.

`position`
: Pointer to the position result.

`findFirst`
: If set to `true`, the first matching element is returned. Use this parameter if the array contains duplicate entries to ensure that the first such entry will be the one returned.

Returns Returns `true` if an exact match was found. In this case, `position` points to the element number where the data was found.

Returns `false` if an exact match was not found. If `false` is returned, `position` points to the element number where the data should be inserted if it was to be added to the array in sorted order.

Comments The array must be sorted in ascending order prior to the search. Use `SysInsertionSort()` or `SysQSort()` to sort the array.

The search starts at element 0 and ends at element (`numOfElements` - 1).

The search function's (`searchF`) prototype is:

```
Int16 _searchF (void const *searchData,
void const *arrayData, Int32 other);
```

The first parameter is the data for which to search, the second parameter is a pointer to an element in the array, and the third parameter is any other necessary data.

The function returns:

- > 0 if the search data is greater than the element
- < 0 if the search data is less than the element
- 0 if the search data is the same as the element

Compatibility Implemented only if 2.0 New Feature Set is present. Prior to Palm OS Garnet version 5.3SC, the *`numOfElements`* parameter was declared to be an `Int16`, limiting the number of elements to be searched to 32,767.

SysBroadcastActionCode Function

Purpose Send the specified action code (launch code) and parameter block to the latest version of every UI application.

Declared In `SystemMgr.h`

Prototype

```
Err SysBroadcastActionCode (UInt16 cmd,
   MemPtr cmdPBP)
```

Parameters

`cmd`
: Action code to send.

`cmdPBP`
: Action code parameter block to send.

Returns Returns 0 if no error, or one of the following errors: `sysErrParamErr`, `memErrNotEnoughSpace`, or `sysErrOutOfOwnerIDs`.

See Also `SysAppLaunch()`, Chapter 2, "Application Startup and Stop." of the *Palm OS Programmer's Companion*, vol. I

SysCopyStringResource Function

Purpose Copy a resource string to a passed string.

Declared In `SysUtils.h`

Prototype

```
void SysCopyStringResource (Char *string,
   Int16 theID)
```

Parameters

`string`
: String to copy the resource string to.

`theID`
: Resource string ID.

Returns Stores a copy of the resource string in `string`.

SysCreateDataBaseList Function

Purpose Generate a list of databases found on the memory cards matching a specific type and return the result. If `lookupName` is `true` then a name in a `tAIN` resource is used instead of the database's name and the list is sorted. Only the last version of a database is returned. Databases with multiple versions are listed only once.

Declared In `SystemMgr.h`

Prototype

```
Boolean SysCreateDataBaseList (UInt32 type,
   UInt32 creator, UInt16 *dbCount,
   MemHandle *dbIDs,
   Boolean lookupName)
```

Parameters

`type`
: The type of database to find. Use `0` to find all databases.

`creator`
: The creator ID of the database to find. Use `0` to find any creator ID.

`dbCount`
: A pointer to an integer value that is updated by this function to the number of matching databases.

`dbIDs`
: A pointer to a handle that gets allocated to contain the database list. Upon return, this references an array of `SysDBListItemType` structures. See the Comments section below for more information.

`lookupName`
: If `true`, `SysCreateDatabaseList` uses `tAIN` names and sorts the list.

Returns

Returns `false` if no databases were found, and `true` if any databases were found. The value of `dbCount` is updated to reflect the number of databases that were found. If at least one database is found, `dbIDs` is updated to reference a array of `SysDBListItemType` structures; this array contains `dbCount` items.

Comments

This function creates a list of unique databases, where unique is defined as having a different type and creator ID. Two or more databases with the same type and creator ID are counted as one. Thus, you cannot use `SysCreateDataBaseList` to build a list of databases that share a common type and creator. There are two exceptions to this rule, however. If `type` is 0 or if `creator` is not 0, the code that removes "non-unique" databases isn't run. It also isn't run if the type is `sysFileTpqa`, since web-clipping databases all have the same type and creator ID.

If this function returns `true` and the value of `dbCount` is greater then 0, than you can iterate through the list of database items, as shown in Listing 54.1

Listing 54.1 Using the SysCreateDatabaseList function

```
SysDBListItemType          *dbListIDsP;
MemHandle                  dbListIDsH;
UInt16                     dbCount = 0;
Boolean                    status;
UInt16                     counter;
SysDBListItemType          theItem;
```

```
status = SysCreateDatabaseList(sysFileTpqa, 0,
                               &dbCount, &dbListIDsH, false);

if (status == true && dbCount > 0)
   {
   dbListIDsP = MemHandleLock (dbListIDsH);
   for (counter = 0; counter < dbCount; counter++)
      if StrCompare(dbListIDsP[counter].name,
                                                "MINE") == 0
      // we found my database
   ...
...
   MemPtrFree (dbListIDsP);
   }
```

NOTE: It is your responsibility to free the memory allocated by this function for the list of databases.

Compatibility Implemented only if 2.0 New Feature Set is present.

SysCreatePanelList Function

Purpose Generate a list of panels found on the memory cards and return the result. Multiple versions of a panel are listed once.

Declared In `SystemMgr.h`

Prototype

```
Boolean SysCreatePanelList (UInt16 *panelCount,
   MemHandle *panelIDs)
```

Parameters

`panelCount`
: Pointer to set to the number of panels.

`panelIDs`
: A pointer to a handle that gets allocated to contain the panel list. Upon return, this references an array of `SysDBListItemType` structures.

Returns Returns `false` if no panels were found, and `true` if any panels were found. The value of `panelCount` is updated to reflect the number of panels that were found. If at least one panel is found, `panelIDs` is updated to reference a array of `SysDBListItemType` structures; this array contains `panelCount` items.

Comments If this function returns `true` and the value of `panelCount` is greater than 0, than you can iterate through the list of panel items, as shown in Listing 54.1. It is your responsibility to free the memory allocated for the panel list.

Compatibility Implemented only if 2.0 New Feature Set is present.

SysCurAppDatabase Function

Purpose Return the card number and database ID of the current application's resource database.

Declared In `SystemMgr.h`

Prototype

```
Err SysCurAppDatabase (UInt16 *cardNoP,
   LocalID *dbIDP)
```

Parameters

`cardNoP`
Pointer to the card number; 0 or 1.

`dbIDP`
Pointer to the database ID.

Returns Returns 0 if no error, or `SysErrParamErr` if an error occurs.

See Also SysAppLaunch(), SysUIAppSwitch()

SysErrString Function

Purpose Returns text to describe an error number. This routine looks up the textual description of a system error number in the appropriate List resource and creates a string that can be used to display that error.

The actual string will be of the form: "<error message> (XXXX)" where XXXX is the hexadecimal error number.

This routine looks for a resource of type 'tstl' and resource ID of (err>>8). It then grabs the string at index (err & 0x00FF) out of that resource.

The first string in the resource is called index #1 by Constructor, NOT #0. For example, an error code of 0x0101 will fetch the first string in the resource.

Declared In `SysUtils.h`

Prototype

```
Char *SysErrString (Err err, Char *strP,
    UInt16 maxLen)
```

Parameters

`err`
: Error number

`strP`
: Pointer to space to form the string

`maxLen`
: Size of `strP` buffer.

Returns Stores the error number string.

Compatibility Implemented only if 2.0 New Feature Set is present.

SysFormPointerArrayToStrings Function

Purpose Form an array of pointers to strings in a block. Useful for setting the items of a list.

Declared In `SysUtils.h`

Prototype

```
MemHandle SysFormPointerArrayToStrings (Char *c,
    Int16 stringCount)
```

Parameters

`c`
: Pointer to packed block of strings, each terminated by a null character.

`stringCount`
Count of strings in block.

Returns Unlocked handle to allocated array of pointers to the strings in the passed block. The returned array points to the strings in the passed packed block. Note that you'll need to free the returned handle when you no longer need it.

See Also `LstSetListChoices()`

sysFtrNumProcessorIs68K Macro

Purpose Determines whether or not the underlying processor is part of the 68K family.

Declared In `SystemMgr.h`

Prototype

```
#define sysFtrNumProcessorIs68K(x)
```

Parameters → *x*
Processor type obtained from a call to `FtrGet()`.

Returns Returns `true` if the underlying processor is a 68K, `false` otherwise.

Comments This macro is typically used in conjunction with `PceNativeCall()`.

Example

```
UInt32 processorType;

FtrGet(sysFileCSystem, sysFtrNumProcessorID, &processorType);
if (sysFtrNumProcessorIs68K(processorType)){
    // processor is 68K
} else {
    // processor is not 68K
}
```

sysFtrNumProcessorIsARM Macro

Purpose Determines whether or not the underlying processor is part of the ARM family.

Declared In `SystemMgr.h`

Prototype

```
#define sysFtrNumProcessorIsARM(x)
```

Parameters → *x*

Processor type obtained from a call to `FtrGet()`.

Returns Returns `true` if the underlying processor is an ARM core, `false` otherwise.

Comments This macro is typically used in conjunction with `PceNativeCall()`.

Example

```
UInt32 processorType;

FtrGet(sysFileCSystem, sysFtrNumProcessorID, &processorType);
if (sysFtrNumProcessorIsARM(processorType)){
    // processor is ARM
} else {
    // processor is not ARM
}
```

SysGetOSVersionString Function

Purpose Return the version number of the Palm™ operating system.

Declared In `SystemMgr.h`

Prototype

```
Char *SysGetOSVersionString()
```

Parameters `None.`

Returns Returns a string such as "v. 3.0."

Comments You must free the returned string using the `MemPtrFree()` function.

Compatibility Implemented only if 3.0 New Feature Set is present.

SysGetROMToken Function

Purpose Return from ROM a value specified by token.

Declared In `SystemMgr.h`

Prototype

```
Err SysGetROMToken (UInt16 cardNo, UInt32 token,
   UInt8 **dataP, UInt16 *sizeP)
```

Parameters

→ *cardNo*
The card on which the ROM to be queried resides. Currently, no Palm hardware provides multiple cards, so this value must be 0.

→ *token*
The value to retrieve, as specified by one of the following tokens:

`sysROMTokenSnum`
The serial number of the ROM, expressed as a text string with no null terminator.

← *dataP*
Pointer to a text buffer that holds the requested value when the function returns.

← *sizeP*
The number of bytes in the `dataP` buffer.

Returns Returns the requested value if no error, or an error code if an error occurs. If this function returns an error, or if the returned pointer to the buffer is `NULL`, or if the first byte of the text buffer is `0xFF`, then no serial number is available.

Comments The serial number is shown to the user in the Application Launcher, along with a checksum digit you can use to validate input when your users read the ID from their device and type it in or tell it to someone else.

Compatibility Implemented only if 3.0 New Feature Set is present. Serial numbers are available only on flash ROM-based units.

See Also "Retrieving the ROM Serial Number" section in the *Palm OS Programmer's Companion*, vol. I shows how to retrieve the ROM serial number and calculate its associated checksum.

SysGetStackInfo Function

Purpose Return the start and end of the current thread's stack.

Declared In `SystemMgr.h`

Prototype

```
Boolean SysGetStackInfo (MemPtr *startPP,
   MemPtr *endPP)
```

Parameters

`startPP`
Upon return, points to the start of the stack.

`endPP`
Upon return, points to the end of the stack.

Returns Returns `true` if the stack has not overflowed, that is, the value of the stack overflow address has not been changed. Returns `false` if the stack overflow value has been overwritten, meaning that a stack overflow has occurred.

Compatibility Implemented only if 3.0 New Feature Set is present.

SysGetTrapAddress Function

Purpose Return the address of a function given its system trap.

Declared In `SystemMgr.h`

Prototype

```
void *SysGetTrapAddress (UInt16 trapNum)
```

Parameters

→ *`trapNum`*
One of the routine selectors defined in `SysTraps.h` (`sysTrap...`) or `CoreTraps.h` on Palm OS version 3.5 and higher.

Returns Returns the address of the corresponding function. Returns the address of the `sysTrapSysUnimplemented` routine if an invalid routine selector is passed; compare the results of this function to `SysGetTrapAddress(sysTrapSysUnimplemented)` if you need to check for an invalid routine selector.

Comments Use this function for performance reasons. You can then use the address it returns to call the function without having to go through the trap dispatch table. This function is mostly useful for optimizing the performance of functions called in a tight loop.

The Palm OS trap dispatch mechanism allows the trap table entries to be modified at any time, either as the result of a system update or a hack. For this reason, it's important to call this function *immediately before* entering the tight loop. If the trap address changes in between when you call `SysGetTrapAddress` and you use the address, the wrong function will be called.

Compatibility On Palm OS 3.0 and earlier, this function contains a bug that causes it to return a garbage value if an invalid routine selector is passed. To prevent this bug from affecting your application, use `SysGlueGetTrapAddress` in the PalmOSGlue library instead of calling this function directly. For more information, see Chapter 80, "PalmOSGlue Library."

SysGremlins Function

Purpose Query the Gremlins facility. You pass a selector for a function and parameters for that function. Gremlins performs the function call and returns the result.

Declared In `SysUtils.h`

Prototype

```
UInt32 SysGremlins (GremlinFunctionType selector,
    GremlinParamsType *params)
```

Parameters

`selector`
: The selector for a function to pass to Gremlins.

`params`
: Pointer to a parameter block used to pass parameters to the function specified by `selector`.

Returns Returns the result of the function performed in Gremlins.

Comments Currently, only one selector is defined, `GremlinIsOn`, which takes no parameters. `GremlinIsOn` returns 0 if Gremlins is not running, non-zero if it is running.

Currently, non-zero values are returned only from the version of Gremlins in the Palm OS emulator. The Gremlins running in the Simulator on a Macintosh and over the serial line via the Palm Debugger return zero for `GremlinIsOn`.

Use this function if you need to alter the application's behavior when Gremlins is running. For example, the debug 3.0 ROM refuses

to bring up the digitizer panel when Gremlins is running under the emulator.

Compatibility Implemented only if 3.0 New Feature Set is present.

In Palm OS 3.2 and later, `SysGremlins` is replaced by the functions defined in the file `HostControl.h`. Specifically, the one selector supported by `SysGremlins` is replaced with the function `HostGremlinIsRunning`. For backward compatibility, `SysGremlins` is mapped to `HostGremlinIsRunning`.

SysHandleEvent Function

Purpose Handle defaults for system events such as hard and soft key presses.

Declared In `SystemMgr.h`

Prototype

```
Boolean SysHandleEvent (EventPtr eventP)
```

Parameters

`eventP`
Pointer to an event.

Returns Returns `true` if the system handled the event.

Comments Applications should call this routine immediately after calling `EvtGetEvent()` unless they want to override the default system behavior. However, overriding the default system behavior is almost never appropriate for an application.

See Also `EvtProcessSoftKeyStroke()`, `KeyRates()`

SysInsertionSort Function

Purpose Sort elements in an array according to the passed comparison function.

Declared In `SysUtils.h`

Prototype

```
void SysInsertionSort (void *baseP,
   UInt16 numOfElements, Int16 width,
   CmpFuncPtr comparF, Int32 other)
```

Parameters

`baseP`
Base pointer to an array of elements.

`numOfElements`
: Number of elements to sort (must be at least 2).

`width`
: Width of an element.

`comparF`
: Comparison function. See Comments, below.

`other`
: Other data passed to the comparison function.

Returns Returns nothing.

Comments Only elements which are out of order move. Moved elements are moved to the end of the range of equal elements. If a large amount of elements are being sorted, try to use the quick sort (see `SysQSort()`).

This is the insertion sort algorithm: Starting with the second element, each element is compared to the preceding element. Each element not greater than the last is inserted into sorted position within those already sorted. A binary search for the insertion point is performed. A moved element is inserted after any other equal elements.

In order to use `SysInsertionSort` you must write a comparison function with the following prototype:

```
Int16 comparF (void *p1, void *p2, Int32 other)
```

Your comparison function must return zero if `p1` equals `p2`, a positive integer value if `p1` is greater than `p2`, and a negative integer value if `p1` is less than `p2`. Note that the value of the parameter named `other` is passed through from the `SysInsertionSort` call and can be used to control the behavior of the `comparF` function if appropriate for your application.

Compatibility The Palm OS 2.0 comparison function (`comparF`) has this prototype:

```
Int comparF (VoidPtr, VoidPtr, Long other);
```

The Palm OS 1.0 comparison function (`comparF`) has this prototype:

```
Int comparF (BytePtr A, BytePtr B, Long other);
```

Prior to Palm OS Garnet version 5.3SC, the `numOfElements` parameter was declared to be an `Int16`, limiting the number of elements to be sorted to 32,767.

See Also `SysQSort()`

SysKeyboardDialog Function

Purpose Pop up the system keyboard if there is a field object with the focus. The field object's text chunk is edited directly.

Declared In `Keyboard.h`

Prototype

```
void SysKeyboardDialog (KeyboardType kbd)
```

Parameters

`kbd`
The keyboard type. See `Keyboard.h`.

Returns Returns nothing. Changes the field's text chunk.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `SysKeyboardDialogV10()`, `FrmSetFocus()`

SysKeyboardDialogV10 Function

Purpose Pop up the system keyboard if there is a field object with the focus. The field object's text chunk is edited directly.

Declared In `Keyboard.h`

Prototype

```
void SysKeyboardDialogV10 ()
```

Parameters `None.`

Returns Returns nothing. The field's text chunk is changed.

Compatibility Corresponds to the 1.0 implementation of `SysKeyboardDialog`.

See Also `SysKeyboardDialog()`, `FrmSetFocus()`

SysLibFind Function

Purpose Return a reference number for a library that is already loaded, given its name.

Declared In `SystemMgr.h`

Prototype

```
Err SysLibFind (const Char *nameP,
   UInt16 *refNumP)
```

Parameters

`nameP`
Pointer to the name of a loaded library.

`refNumP`
Pointer to a variable for returning the library reference number (on failure, this variable is undefined)

Returns 0 if no error; otherwise: `sysErrLibNotFound` (if the library is not yet loaded), or another error returned from the library's install entry point.

Comments Most built-in libraries (NetLib, serial, IR) are preloaded automatically when the system is reset. Third-party libraries must be loaded before this call can succeed (use `SysLibLoad()`). You can check if a library is already loaded by calling `SysLibFind` and checking for a 0 error return value (it will return a non-zero value if the library is not loaded).

SysLibInstall Function

Purpose Installs a library into the Library table and calls the library's install entry point.

Declared In `SystemMgr.h`

Prototype

```
Err SysLibInstall (SysLibEntryProcPtr libraryP,
   UInt16 *refNumP)
```

Parameters

→ *`libraryP`*
Pointer to the library being installed.

← *`refNumP`*
Pointer to the variable in which the installed library reference number is stored upon return.

Returns Returns 0 if no error, `memErrNotEnoughSpace` if an error occurs while allocating memory in the library table, or, if an error is

returned by the call to the library's install entry point, that error is returned and the reference number is set to `sysInvalidRefNum`.

Comments This routine is largely used by the Palm OS, but can also be called by applications that need to install their own libraries.

Compatibility Implemented only if 2.0 New Feature Set is present.

SysLibLoad Function

Purpose Load a library given its database creator and type.

Declared In `SystemMgr.h`

Prototype

```
Err SysLibLoad (UInt32 libType,
   UInt32 libCreator, UInt16 *refNumP)
```

Parameters

`libType`
Type of library database.

`libCreator`
Creator of library database.

`refNumP`
Pointer to variable for returning the library reference number (on failure, `sysInvalidRefNum` is returned in this variable)

Returns 0 if no error; otherwise: `sysErrLibNotFound`, `sysErrNoFreeRAM`, `sysErrNoFreeLibSlots`, or other error returned from the library's install entry point.

Comments When an application no longer needs a library that it *successfully* loaded via `SysLibLoad`, it is responsible for unloading the library by calling `SysLibRemove()` and passing it the library reference number returned by `SysLibLoad`. More information is available in the white paper on shared libraries, which you can find on the Palm developer support web site.

Compatibility Implemented only if 2.0 New Feature Set is present.

SysLibRemove Function

Purpose Unload a library previously loaded with `SysLibLoad()`.

Declared In `SystemMgr.h`

Prototype

```
Err SysLibRemove (UInt16 refNum)
```

Parameters → *refNum*
The library reference number.

Returns 0 if no error; otherwise `sysErrParamErr` if the `refNum` is not a valid library reference number.

Comments `SysLibRemove` releases the dynamic memory allocated to the shared library's dispatch table, resources, and global variables.

SysQSort Function

Purpose Sort elements in an array according to the supplied comparison function.

Declared In `SysUtils.h`

Prototype

```
void SysQSort (void *baseP, UInt16 numOfElements,
   Int16 width, CmpFuncPtr comparF, Int32 other)
```

Parameters

`baseP`
Base pointer to an array of elements.

`numOfElements`
Number of elements to sort (must be at least 2).

`width`
Width of an element.

`comparF`
Comparison function. See Comments, below.

`other`
Other data passed to the comparison function.

Returns Returns nothing.

Comments Equal records can be in any position relative to each other because a quick sort tends to scramble the ordering of records. As a result, calling `SysQSort` multiple times can result in a different order if the records are not completely unique. If you don't want this behavior, use the insertion sort instead (see `SysInsertionSort()`).

To pick the pivot point, the quick sort algorithm picks the middle of three records picked from around the middle of all records. That way, the algorithm can take advantage of partially sorted data.

These optimizations are built in:

- The routine contains its own stack to limit uncontrolled recursion. When the stack is full, an insertion sort is used because it doesn't require more stack space.
- An insertion sort is also used when the number of records is low. This avoids the overhead of a quick sort which is noticeable for small numbers of records.
- If the records seem mostly sorted, an insertion sort is performed to move only those few records that need to be moved.

In order to use `SysQSort` you must write a comparison function with the following prototype:

```
Int16 comparF (void *p1, void *p2, Int32 other)
```

Your comparison function must return zero if `p1` equals `p2`, a positive integer value if `p1` is greater than `p2`, and a negative integer value if `p1` is less than `p2`. Note that the value of the parameter named `other` is passed through from the `SysQSort` call and can be used to control the behavior of the `comparF` function if appropriate for your application.

Compatibility

The Palm OS 2.0 comparison function (`comparF`) has this prototype:

```
Int comparF (VoidPtr, VoidPtr, Long other);
```

The Palm OS 1.0 comparison function (`comparF`) has this prototype:

```
Int comparF (BytePtr A, BytePtr B, Long other);
```

Prior to Palm OS Garnet version 5.3SC the *`numOfElements`* parameter was declared to be an `Int16`, limiting the number of elements to be sorted to 32,767.

See Also

`SysInsertionSort()`

SysRandom Function

Purpose Return a random number anywhere from 0 to `sysRandomMax`.

Declared In `SysUtils.h`

Prototype
```
Int16 SysRandom (Int32 newSeed)
```

Parameters
`newSeed`
New seed value, or 0 to use existing seed.

Returns Returns a random number.

SysReset Function

Purpose Perform a soft reset and reinitialize the globals and the dynamic memory heap.

Declared In `SystemMgr.h`

Prototype
```
void SysReset (void)
```

Parameters `None.`

Returns No return value.

Comments This routine resets the system, reinitializes the globals area and all system managers, and reinitializes the dynamic heap. All database information is preserved. This routine is called when the user presses the hidden reset switch on the device.

When running an application using the simulator, this routine looks for two data files that represent the memory of card 0 and card 1. If these are found, the Palm OS memory image is created using them. If they are not found, they are created.

When running an application on the device, this routine simply looks for the memory cards at fixed locations.

SysSetAutoOffTime Function

Purpose Set the time out value in seconds for auto-power-off. Zero means never power off.

Declared In `SystemMgr.h`

Prototype

```
UInt16 SysSetAutoOffTime (UInt16 seconds)
```

Parameters

`seconds`
Time out in seconds, or 0 for no time out.

Returns Returns previous value of time out in seconds.

SysSetTrapAddress Function

Purpose Set the address of the function corresponding to a system trap.

Declared In `SystemMgr.h`

Prototype

```
Err SysSetTrapAddress (UInt16 trapNum,
    void *procP)
```

Parameters

→ *trapNum*
One of the routine selectors defined in `SysTraps.h` (`sysTrap...`) or `CoreTraps.h` on Palm OS version 3.5 and higher.

→ *procP*
Pointer to a function that the trap identified by `trapNum` is to point to.

Returns Returns 0 if no error, or `SysErrParamErr` if `trapNum` is greater than the number of traps in the trap table.

Comments This function is useful for patching a system trap, in combination with `SysGetTrapAddress()`. To patch a system trap in your application, first call `SysGetTrapAddress` to get the trap address and then save this value somewhere. Next use `SysSetTrapAddress` to set the trap address to point to your function. Before your application exits, remove the patch by calling `SysSetTrapAddress` and passing in the original trap address you saved.

WARNING! If your application patches a system trap using this function, you *must* remove the patch before your application exits. Do *not* use this mechanism to permanently patch system traps as it may cause unpredictable results for the system and other applications.

Compatibility If either 5.0 New Feature Set or Palm OS Cobalt Feature Set is present this function is unimplemented.

SysStringByIndex Function

Purpose Copy a string out of a string list resource by index. String list resources are of type `'tSTL'` and contain a list of strings and a prefix string.

ResEdit always displays the items in the list as starting at 1, not 0. Consider this when creating your string list.

Declared In `SysUtils.h`

Prototype

```
Char *SysStringByIndex (UInt16 resID,
   UInt16 index, Char *strP, UInt16 maxLen)
```

Parameters

`resID`
: Resource ID of the string list.

`index`
: String to get out of the list.

`strP`
: Pointer to space to form the string.

`maxLen`
: Size of `strP` buffer.

Returns Returns a pointer to the copied string. The string returned from this call will be the prefix string appended with the designated index string. Indices are 0-based; index 0 is the first string in the resource.

Compatibility Implemented only if 2.0 New Feature Set is present.

SysTaskDelay Function

Purpose Put the processor into doze mode for the specified number of ticks.

Declared In `SystemMgr.h`

Prototype
```
Err SysTaskDelay (Int32 delay)
```

Parameters

`delay`
Number of ticks to wait (see `SysTicksPerSecond()`)

Returns Returns 0 if no error.

See Also EvtGetEvent()

SysTicksPerSecond Function

Purpose Return the number of ticks per second. This routine allows applications to be tolerant of changes to the ticks per second rate in the system.

Declared In `SystemMgr.h`

Prototype
```
UInt16 SysTicksPerSecond (void)
```

Parameters None

Returns Returns the number of ticks per second.

Compatibility Implemented only if 2.0 New Feature Set is present.

SysUIAppSwitch Function

Purpose Try to make the current UI application quit and then launch the UI application specified by card number and database ID.

Declared In `SystemMgr.h`

Prototype
```
Err SysUIAppSwitch (UInt16 cardNo, LocalID dbID,
   UInt16 cmd, MemPtr cmdPBP)
```

Parameters

→ *cardNo*
Card number for the new application; currently only card 0 is valid.

→ *dbID*
ID of the new application's resource database.

→ *cmd*
Action code (launch code).

→ *cmdPBP*
Action code (launch code) parameter block.

Returns Returns 0 if no error.

May display a fatal error message if the `cardNo` parameter is invalid. On debug ROMs, displays a fatal error message if there is no currently running application.

Comments Do not use this function to open the system-supplied Application Launcher application. If a third-party launch is installed, you'll likely want to launch that one instead. To do this, enqueue a `keyDownEvent` that contains a `launchChr`, as shown in the section "Application Launcher" of the user interface chapter in the *Palm OS Programmer's Companion*, vol. I. This will run whatever is run whenever you tap on the Applications icon.

If you are passing a parameter block (the `cmdPBP` parameter), you must set the owner of the parameter block chunk to the operating system. To do this, and for more information, see `MemPtrSetOwner()`. If the parameter block structure contains references by pointer or handle to any other chunks, you also must set the owner of those chunks by using `MemHandleSetOwner()` or `MemPtrSetOwner()` as appropriate. If you set the owner of this parameter block properly, the system maintains the parameter block and frees it when the second application quits. If you don't set the owner of the parameter block, the system frees the parameter block as soon as the calling application quits, causing unpredictable results.

See Also `SysAppLaunch()`, Chapter 2, "Application Startup and Stop," in the *Palm OS Programmer's Companion*, vol. I.

Application-Defined Functions

PilotMain Function

Purpose The entry point for all Palm OS applications, this function's sole purpose is to receive and respond to launch codes.

Declared In `SystemMgr.h`

Prototype

```
UInt32 PilotMain(UInt16 cmd, void *cmdPBP,
    UInt16 launchFlags)
```

Parameters

→ *cmd*

The launch code to which your application is to respond. See Chapter 1, "Application Launch Codes," on page 3 for a list of predefined launch codes. You may create additional launch codes; see "Creating Your Own Launch Codes" on page 28 of the *Palm OS Programmer's Companion*, vol. I.

→ *cmdPBP*

A pointer to a structure containing any launch-command-specific parameters, or `NULL` if the launch code has none. See the description of each launch code for a description of the parameter structure that accompanies it, if any.

→ *launchFlags*

Flags that indicate whether your application's global variables are available, whether your application is now the active application, whether it already was the active application, and so on. See "Launch Flags" on page 37 for a list of launch flags.

Returns Return `errNone` if your application processed the launch code successfully, or an appropriate error code if there was a problem. When another application invokes your application using `SysAppLaunch()`, this value is returned to the caller.

Comments See Chapter 2, "Application Startup and Stop," on page 19 of the *Palm OS Programmer's Companion*, vol. I for a discussion on how applications receive and handle launch codes, with examples.

55

Text Manager

This chapter provides information about the text manager API declared in `TextMgr.h` by discussing these topics:

- Text Manager Data Structures
- Text Manager Constants
- Text Manager Functions

For more information on the text manager, see the chapter "Localized Applications" on page 385 in the *Palm OS Programmer's Companion*, vol. I.

Text Manager Data Structures

CharEncodingType Typedef

Purpose The `CharEncodingType` enum specifies possible character encodings. The Character Encoding Constants define the possible values for `CharEncodingType` variables.

Prototype

```
UInt8 CharEncodingType;
```

Comments A given handheld supports a single character encoding. The currently available handhelds support either the Palm™ version of Windows code page 1252[1] (an extension of ISO Latin 1), the Palm version of Windows code page 932[1] (an extension of Shift JIS), or GB2312. In addition, Palm OS licensees and some third-party developers provide support for additional character encodings including Big-5, Hebrew, Arabic, Thai, Korean, and Cyrillic.

1. This encoding is identical to its Windows counterpart with some additional characters added in the control range.

Compatibility Prior to version 4.0, `CharEncodingType` was an enum that defined only eight character encodings. The Palm OS® 4.0 definition of `CharEncodingType` is compatible with the previous definition.

TranslitOpType Typedef

Purpose Specifies the transliteration operation to be performed by a given call to `TxtTransliterate()`. Each character encoding contains its own set of special transliteration operations, the values for which begin at `translitOpCustomBase`.

Prototype

```
typedef UInt16 TranslitOpType;
```

Comments The following operations are always available, regardless of the character encoding used on the handheld:

Constant	Value	Description
`translitOpUpperCase`	0	Converts the character to uppercase letters.
`translitOpLowerCase`	1	Converts the characters to lowercase letters.
translitOpReserved2	2	Reserved
translitOpReserved3	3	Reserved
`translitOpPreprocess`	0x8000	A mask that , when OR'd to a transliteration operation value, indicates that `TxtTransliterate` should not perform the operation but should instead return (in `ioDstLength`) the amount of space required for the output text.

TxtConvertStateType Struct

Purpose This structure is used to maintain state across calls to `TxtConvertEncoding()`. It is essentially opaque; simply declare a structure of this type and pass a pointer to your structure when making multiple calls to `TxtConvertEncoding()` for a single source text buffer.

Prototype

```
typedef struct {
  UInt8 ioSrcState[kTxtConvertStateSize];
  UInt8 ioDstState[kTxtConvertStateSize];
} TxtConvertStateType;
```

`kTxtConvertStateSize` is simply a constant that determines the size of the source and destination state buffers.

Text Manager Constants

Character Encoding Constant Modifiers

The following flag can be OR'd with the destination character encoding constant (one of those listed in "Character Encoding Constants" on page 825) that is passed to `TxtConvertEncoding()`.

Constant	Value	Description
`charEncodingDstBestFitFlag`	`0x80`	Causes `TxtConvertEncoding()` to make an extra effort to convert characters in the source encoding to similar (if not equal) characters in the destination encoding.

As an example, when converting from `charEncodingUCS2` to `charEncodingPalmGB`, no mapping exists for `0x00A1` (INVERTED EXCLAMATION MARK), since this character doesn't exist within `charEncodingPalmGB`. In this instance, then `TxtConvertEncoding()` returns `txtErrNoCharMapping` for this mapping. If you OR the `charEncodingDstBestFitFlag` with the destination character encoding, however, `TxtConvertEncoding()` converts the character to `chrExclamationMark` (which is close). Generally, the operating system tries to support as many CP1252 characters as possible in the "best fit" table.

One situation when the best fit flag is commonly used is when you are trying to display text to the user (in which case the destination encoding is the device encoding). Because the best fit flag results in a lossy conversion, you typically don't want to use it on data which

you will write back to storage. But for display, you almost always want to use best fit conversion because it maximizes the amount of meaningful information that may reach the user.

If `charEncodingDstBestFitFlag` is 1 and either the source or destination encoding is unknown, `TxtConvertEncoding()` copies anything that is 7-bit ASCII from the source to the destination. It then returns `txtErrUnknownEncodingFallbackCopy`. The rules for unknown characters apply during this 7-bit copy; if an inconvertible character is encountered the substitution string (if one has been specified) is used in its place and `txtErrNoCharMapping` is returned instead.

Compatibility

The `charEncodingDstBestFitFlag` is only supported in Palm OS versions that support Simplified Chinese. On Palm OS versions that don't support Simplified Chinese, use of this flag will prevent `TxtConvertEncoding()` from recognizing the destination encoding. To safely use the best fit flag on a variety of Palm OS releases, call `TxtGlueConvertEncoding()` instead of `TxtConvertEncoding()`.

Character Encoding Feature Attributes

To obtain the attributes of the character encoding supported by the handheld's ROM, request the value of the `sysFtrNumCharEncodingFlags` feature. You do this by calling `FtrGet()` as follows:

```
err = FtrGet(sysFtrCreator, sysFtrNumEncodingFlags, &value);
```

Use the following bit masks to interpret `value`:

Constant	Value	Description
`charEncodingOnlySingleByte`	`0x00000001`	The character encoding consists only of single-byte characters.
`charEncodingHasDoubleByte`	`0x00000002`	The character encoding contains one or more double-byte characters.

Constant	Value	Description
`charEncodingHasLigatures`	`0x00000004`	The character encoding has ligatures.
`charEncodingRightToLeft`	`0x00000008`	The character encoding supports a writing system that primarily renders text right-to-left.

Character Attributes

The following constants define the flags that identify various character attributes. These replace the old flags defined in `CharAttr.h`, but are bit-compatible.

NOTE: `TextMgr.h` defines a set of functions and macros that can be used determine a character's attributes. These include `TxtCharIsAlpha()`, `TxtCharIsCntrl()`, and `TxtCharIsValid()`.

Constant	Value	Description
`charAttr_XA`	`0x0200`	Extra alphabetic
`charAttr_XS`	`0x0100`	Extra space
`charAttr_BB`	`0x0080`	BEL, BS, etc.
`charAttr_CN`	`0x0040`	CR, FF, HT, NL, VT
`charAttr_DI`	`0x0020`	'0'-'9'
`charAttr_LO`	`0x0010`	'a'-'z' and lowercase extended characters
`charAttr_PU`	`0x0008`	Punctuation
`charAttr_SP`	`0x0004`	Space

Constant	Value	Description
`charAttr_UP`	`0x0002`	'A'-'Z' and uppercase extended characters
`charAttr_XD`	`0x0001`	'0'-'9', 'A'-'F', 'a'-'f'

These character attribute flags can be combined in various ways. `TextMgr.h` defines a number of standard combinations:

Constant	Value	Description
`charAttrPrint`	`(charAttr_DI|charAttr_LO|charAttr_PU|charAttr_SP|charAttr_UP|charAttr_XA)`	Printable characters
`charAttrSpace`	`(charAttr_CN|charAttr_SP|charAttr_XS)`	Whitespace characters
`charAttrAlNum`	`(charAttr_DI|charAttr_LO|charAttr_UP|charAttr_XA)`	Alphanumeric characters
`charAttrAlpha`	`(charAttr_LO|charAttr_UP|charAttr_XA)`	Alphabetic characters
`charAttrCntrl`	`(charAttr_BB|charAttr_CN)`	Control characters
`charAttrGraph`	`(charAttr_DI|charAttr_LO|charAttr_PU|charAttr_UP|charAttr_XA)`	Printable, non-space characters
`charAttrDelim`	`(charAttr_SP|charAttr_PU)`	Delimiters

Character Sizes

These constants can be useful when working with characters of varying sizes.

Constant	Value	Description
`sizeOf7BitChar`	`1`	The size, in bytes, of a 7-bit character. In C, characters are treated like integers: `sizeof(0x0D) == 2` (as is `sizeof('a')`, `sizeof('\0')`, and `sizeof(chrNull)`). Because of this, it's safest to use the `sizeOf7BitChar` macro to document buffer size and string length calculations. Note that this can only be used with low-ASCII characters, as anything else might be the high byte of a double-byte character.
`maxCharBytes`	`3`	Maximum size a single WChar character will occupy in a text string.

Text Manager Functions

TxtByteAttr Function

Purpose Return the possible locations of a given byte within a multi-byte character.

Declared In `TextMgr.h`

Prototype

```
UInt8 TxtByteAttr (UInt8 inByte)
```

Parameters → *`inByte`*
A byte representing all or part of a valid character.

Returns Returns a byte with one or more of the following bits set:

`byteAttrFirst`	First byte of multi-byte character.
`byteAttrLast`	Last byte of multi-byte character.
`byteAttrMiddle`	Middle byte of multi-byte character.
`byteAttrSingle`	Single-byte character.

Comments If `inByte` is valid in more than one location of a character, multiple return bits are set. For example, 0x40 in the Shift JIS character encoding is valid as a single-byte character and as the low-order byte of a double-byte character. Thus, the return value for `TxtByteAttr(0x40)` on a Shift JIS system has both the

`byteAttrSingle` and `byteAttrLast` bits set. `TextMgr.h` defines two constants that represent double-byte encoding combinations:

byteAttrSingleLow	Either a single-byte character or the low-order byte of a double-byte character.
byteAttrHighLow	Either the first byte of a multi-byte character or the last byte of a multi-byte character.

Every byte in a stream of double-byte data must be either a single byte, a single/low byte (`byteAttrSingleLow`), or a high/low byte (`byteAttrHighLow`).

Text manager functions that need to determine the byte positioning of a character use `TxtByteAttr` to do so. You rarely need to use this function yourself.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueByteAttr`. For more information, see Chapter 80, "PalmOSGlue Library."

TxtCaselessCompare Function

Purpose Perform a case-insensitive comparison of two text buffers.

Declared In `TextMgr.h`

Prototype

```
Int16 TxtCaselessCompare (const Char *s1,
   UInt16 s1Len, UInt16 *s1MatchLen,
   const Char *s2,
   UInt16 s2Len, UInt16 *s2MatchLen)
```

Parameters

→ *s1*
Pointer to the first text buffer to compare.

→ *s1Len*
Length in bytes of the text pointed to by `s1`.

← *s1MatchLen*
Points to the offset of the first character in `s1` that determines the sort order. Pass `NULL` for this parameter if you don't need to know this number.

→ *s2*
Pointer to the second text buffer to compare.

→ *s2Len*
Length in bytes of the text pointed to by `s2`.

← *s2MatchLen*
Points to the offset of the first character in `s2` that determines the sort order. Pass `NULL` for this parameter if you don't need to know this number.

Returns Returns one of the following values:

< 0 If `s1` occurs before `s2` in alphabetical order.

> 0 If `s1` occurs after `s2` in alphabetical order.

0 If the two substrings that were compared are equal.

Comments In certain character encodings (such as Shift JIS), one character may be accurately represented as either a single-byte character or a multi-byte character. `TxtCaselessCompare` accurately matches a single-byte character with its multi-byte equivalent. For this reason, the values returned in `s1MatchLen` and `s2MatchLen` are not always equal.

You must make sure that the parameters `s1` and `s2` point to a the start of a valid character. That is, they must point to the first byte of a multi-byte character or they must point to a single-byte character; if they don't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCaselessCompare`. For more information, see Chapter 80, "PalmOSGlue Library."

In Palm OS 4.0, the `TxtCaselessCompare` function terminates when it finds a null byte in the string. In earlier releases, it terminated only when it reached the ending byte specified by the length parameters.

See Also `StrCaselessCompare()`, `TxtCompare()`, `StrCompare()`

TxtCharAttr Function

Purpose Return a character's attributes.

Declared In `TextMgr.h`

Prototype

```
UInt16 TxtCharAttr (WChar inChar)
```

Parameters

→ *inChar*
Any valid character.

Returns Returns a 16-bit unsigned value with any of the following bits set:

`charAttrPrint`	Printable
`charAttrSpace`	Blank space, tab, or newline
`charAttrAlNum`	Alphanumeric
`charAttrAlpha`	Alphabetic
`charAttrCntrl`	Control character
`charAttrGraph`	Character that appears on the screen; that is, is not whitespace, a control character, or a virtual character.
`charAttrDelim`	Word delimiter (whitespace or punctuation).

Comments The character passed to this function must be a valid character given the system encoding.

This function is used in the text manager's character attribute macros (`TxtCharIsAlNum()`, `TxtCharIsCntrl()`, and so on). The macros perform operations analogous to the standard C functions `isPunct`, `isPrintable`, and so on. Usually, you'd use one of these macros instead of calling `TxtCharAttr` directly.

To obtain attributes specific to a given character encoding, use `TxtCharXAttr()`.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharAttr`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `TxtCharIsValid()`

TxtCharBounds Function

Purpose Return the boundaries of a character containing the byte at a specified offset in a string.

Declared In `TextMgr.h`

Prototype

```
WChar TxtCharBounds (const Char *inText,
   UInt32 inOffset, UInt32 *outStart,
   UInt32 *outEnd)
```

Parameters

→ *inText*
Pointer to the text buffer to search.

→ *inOffset*
A valid offset into the buffer `inText`. This location may contain a byte in any position (start, middle, or end) of a multi-byte character.

← *outStart*
Points to the starting offset of the character containing the byte at `inOffset`.

← *outEnd*
Points to the ending offset of the character containing the byte at `inOffset`.

Returns Returns the character located between the offsets `outStart` and `outEnd`.

Comments Use this function to determine the boundaries of a character in a string or text buffer.

If the byte at `inOffset` is valid in more than one location of a character, the function must search back toward the beginning of the text buffer until it finds an unambiguous byte to determine the appropriate boundaries. For this reason, `TxtCharBounds` is often slow and should be used only where needed.

You must make sure that the parameter `inText` points to the beginning of the string. That is, if the string begins with a multi-byte character, `inText` must point to the first byte of that character; if it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharBounds`. For more information, see Chapter 80, "PalmOSGlue Library."

TxtCharEncoding Function

Purpose Return the minimum encoding required to represent a character.

Declared In `TextMgr.h`

Prototype

```
CharEncodingType TxtCharEncoding (WChar inChar)
```

Parameters

→ *inChar*
A valid character.

Returns A `CharEncodingType` value that indicates the minimum encoding required to represent `inChar`. If the character isn't recognizable, `charEncodingUnknown` is returned.

Comments The minimum encoding is the encoding that represents the fewest number of characters while still containing the character specified in `inChar`. For example, if the character is a blank or a tab character, the minimum encoding is `charEncodingAscii` because these characters can be represented in single-byte ASCII. If the character is a ü, the minimum encoding is `charEncodingISO8859_1`.

This function is used by `TxtStrEncoding()`, which is the function that most applications should use to determine the character encoding for tagging text (for instance, for email).

Use `TxtMaxEncoding()` to determine the order of encodings.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharEncoding`. For more information, see Chapter 80, "PalmOSGlue Library."

Palm OS only supports a single character encoding at a time. Because of this, the result of `TxtCharEncoding` is always logically equal to or less than the encoding used on the current system. That is, you'll only receive a return value of `charEncodingISO8859_1` if you're running on a US or European system and you pass a non-ASCII character.

See Also TxtStrEncoding(), TxtMaxEncoding()

TxtCharIsAlNum Macro

Purpose Indicates if the character is alphanumeric.

Declared In `TextMgr.h`

Prototype

```
#define TxtCharIsAlNum (ch)
```

Parameters → *ch*

A valid character.

Returns Returns `true` if the character is a letter in an alphabet or a numeric digit, `false` otherwise.

Compatibility Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsAlNum`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `TxtCharIsDigit()`, `TxtCharIsAlpha()`

TxtCharIsAlpha Macro

Purpose Indicates if a character is a letter in an alphabet.

Declared In `TextMgr.h`

Prototype

```
#define TxtCharIsAlpha (ch)
```

Parameters → *ch*

A valid character.

Returns Returns `true` if the character is a letter in an alphabet, `false` otherwise.

Compatibility Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with

the PalmOSGlue library and call `TxtGlueCharIsAlpha`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also TxtCharIsAlNum(), TxtCharIsLower(), TxtCharIsUpper()

TxtCharIsCntrl Macro

Purpose Indicates if a character is a control character.

Declared In `TextMgr.h`

Prototype

```
#define TxtCharIsCntrl (ch)
```

Parameters → *ch*
A valid character.

Returns Returns `true` if the character is a non-printable character, such as the bell character or a carriage return; `false` otherwise.

Compatibility Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsCntrl`. For more information, see Chapter 80, "PalmOSGlue Library."

TxtCharIsDelim Macro

Purpose Indicates if a character is a delimiter.

Declared In `TextMgr.h`

Prototype

```
#define TxtCharIsDelim (ch)
```

Parameters → *ch*
A valid character.

Returns Returns `true` if the character is a word delimiter (whitespace or punctuation), `false` otherwise.

Compatibility Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsDelim`. For more information, see Chapter 80, "PalmOSGlue Library."

TxtCharIsDigit Macro

Purpose Indicates if the character is a decimal digit.

Declared In `TextMgr.h`

Prototype
```
#define TxtCharIsDigit (ch)
```

Parameters → *ch*
A valid character.

Returns Returns `true` if the character is 0 through 9, `false` otherwise.

Compatibility Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsDigit`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `TxtCharIsAlNum()`, `TxtCharIsHex()`

TxtCharIsGraph Macro

Purpose Indicates if a character is a graphic character.

Declared In `TextMgr.h`

Prototype
```
#define TxtCharIsGraph (ch)
```

Parameters → *ch*
A valid character.

Returns Returns `true` if the character is a graphic character, `false` otherwise.

Comments A graphic character is any character visible on the screen, in other words, letters, digits, and punctuation marks. A blank space is not a graphic character because it is not visible.

This macro differs from `TxtCharIsPrint()` in that it returns `false` if the character is whitespace. `TxtCharIsPrint` returns `true` if the character is whitespace.

Compatibility Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsGraph`. For more information, see Chapter 80, "PalmOSGlue Library."

TxtCharIsHardKey Macro

Purpose Returns true if the character is one of the hard keys on the device.

Declared In `TextMgr.h`

Prototype

```
#define TxtCharIsHardKey (m, ch)
```

Parameters

→ *m*
The modifier keys from the `keyDownEvent`.

→ *ch*
The character from the `keyDownEvent`.

Returns `true` if the character is one of the built-in hard keys on the device, `false` otherwise.

Compatibility Valid only if International Feature Set is present.

See Also `ChrIsHardKey()`

TxtCharIsHex Macro

Purpose Indicates if a character is a hexadecimal digit.

Declared In `TextMgr.h`

Prototype

```
#define TxtCharIsHex (ch)
```

Parameters

→ *ch*
A valid character.

Returns Returns `true` if the character is a hexadecimal digit from 0 to F, `false` otherwise.

Compatibility Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsHex`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `TxtCharIsDigit()`

TxtCharIsLower Macro

Purpose Indicates if a character is a lowercase letter.

Declared In `TextMgr.h`

Prototype
```
#define TxtCharIsLower (ch)
```

Parameters → *ch*
A valid character.

Returns Returns `true` if the character is a lowercase letter, `false` otherwise.

Compatibility Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsLower`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `TxtCharIsAlpha()`, `TxtCharIsUpper()`

TxtCharIsPrint Macro

Purpose Indicates if a character is printable.

Declared In `TextMgr.h`

Prototype
```
#define TxtCharIsPrint (ch)
```

Parameters → *ch*
A valid character.

Returns Returns `true` if the character is not a control character, `false` otherwise.

Comments This macro differs from `TxtCharIsGraph()` in that it returns `true` if the character is whitespace. `TxtCharIsGraph` returns `false` if the character is whitespace.

If you are using a debug ROM and you pass a virtual character to this macro, a fatal alert is generated.

Compatibility Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsPrint`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `TxtCharIsValid()`

TxtCharIsPunct Macro

Purpose Indicates if a character is a punctuation mark.

Declared In `TextMgr.h`

Prototype

```
#define TxtCharIsPunct (ch)
```

Parameters → *ch*

A valid character.

Returns Returns `true` if the character is a punctuation mark, `false` otherwise.

Compatibility Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsPunct`. For more information, see Chapter 80, "PalmOSGlue Library."

TxtCharIsRockerKey Macro

Purpose Determines whether or not a given character and event modifier combination indicate that the key is a 5-way rocker key.

Declared In `TextMgr.h`

Prototype

```
#define TxtCharIsRockerKey (m, c)
```

Parameters → *m*

The event modifier.

→ *c*

The character.

Returns `true` if *c* is one of the a 5-way rocker keys. *c* is a rocker key if the event modifier *m* has the command bit set and *c* is in the proper range:

`vchrRockerUp` <= *c* <= `vchrRockerCenter`

TxtCharIsSpace Macro

Purpose Indicates if a character is a whitespace character.

Declared In `TextMgr.h`

Prototype

```
#define TxtCharIsSpace (ch)
```

Parameters → *ch*
A valid character.

Returns Returns `true` if the character is whitespace such as a blank space, tab, or newline; `false` otherwise.

Compatibility Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsSpace`. For more information, see Chapter 80, "PalmOSGlue Library."

TxtCharIsUpper Macro

Purpose Indicates if a character is an uppercase letter.

Declared In `TextMgr.h`

Prototype

```
#define TxtCharIsUpper (ch)
```

Parameters → *ch*
A valid character.

Returns Returns `true` if the character is an uppercase letter, `false` otherwise.

Compatibility Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsUpper`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `TxtCharIsAlpha()`, `TxtCharIsLower()`

TxtCharIsValid Function

Purpose Determine whether a character is valid character given the Palm OS character encoding.

Declared In `TextMgr.h`

Prototype
```
Boolean TxtCharIsValid (WChar inChar)
```

Parameters → *inChar*
A character.

Returns Returns `true` if `inChar` is a valid character; `false` if `inChar` is not a valid character.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsValid`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `TxtCharAttr()`, `TxtCharIsPrint()`

TxtCharSize Function

Purpose Return the number of bytes required to store the character in a string.

Declared In `TextMgr.h`

Prototype
```
UInt16 TxtCharSize (WChar inChar)
```

Parameters → *inChar*
A valid character.

Returns The number of bytes required to store the character in a string.

Comments Although character variables are always two-byte long `WChar` values, in some character encodings such as Shift-JIS, characters in strings are represented by a mix of one or more bytes per character. If the character can be represented by a single byte (its high-order byte is 0), it is stored in a string as a single-byte character.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharSize`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `TxtCharBounds()`

TxtCharWidth Function

Purpose Return the width required to display the specified character in the current font. If the specified character does not exist within the current font, the missing character symbol is substituted.

Declared In `TextMgr.h`

Prototype

```
Int16 TxtCharWidth (WChar inChar)
```

Parameters → *inChar*
A valid character.

Returns Returns the width of the specified character (in pixels).

Comments Use `FntWCharWidth()` or `FntGlueWCharWidth` instead of this routine.

Compatibility Implemented only if International Feature Set is present.

TxtCharXAttr Function

Purpose Return the extended attribute bits for a character.

Declared In `TextMgr.h`

Prototype

```
UInt16 TxtCharXAttr (WChar inChar)
```

Parameters → *inChar*
A valid character.

Returns Returns an unsigned 16-bit value with one or more extended attribute bits set. For specific return values, look in the header files that are specific to certain character encodings (`CharLatin.h` or `CharShiftJIS.h`).

Comments To interpret the results, you must know the character encoding being used. Use `FtrGet()` with `sysFtrNumEncoding` as the feature number to determine the character encoding. This returns one of the `CharEncodingType` values. For example:

```
WChar ch;
UInt16 attr;
UInt32 encoding;
...
attr = TxtCharXAttr(ch);
if (FtrGet(sysFtrCreator, sysFtrNumEncoding,
  &encoding) != errNone)
```

```
  encoding = charEncodingPalmLatin;
if (encoding == charEncodingUTF8) {
}
```

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharXAttr`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `TxtCharAttr()`

TxtCompare Function

Purpose Performs a case-sensitive comparison of all or part of two text buffers.

Declared In `TextMgr.h`

Prototype

```
Int16 TxtCompare (const Char *s1, UInt16 s1Len,
   UInt16 *s1MatchLen, const Char *s2,
   UInt16 s2Len,
   UInt16 *s2MatchLen)
```

Parameters

→ *s1*
Pointer to the first text buffer to compare.

→ *s1Len*
The length in bytes of the text pointed to by `s1`.

← *s1MatchLen*
Points to the offset of the first character in `s1` that determines the sort order. Pass `NULL` for this parameter if you don't need to know this number.

→ *s2*
Pointer to the second text buffer to compare.

→ *s2Len*
The length in bytes of the text pointed to by `s2`.

← *s2MatchLen*
Points to the offset of the first character in `s2` that determines the sort order. Pass `NULL` for this parameter if you don't need to know this number.

Returns Returns one of the following values:

< 0	If `s1` occurs before `s2` in alphabetical order.
> 0	If `s1` occurs after `s2` in alphabetical order.
0	If the two substrings that were compared are equal.

Comments

This function performs a case-sensitive comparison. If you want to perform a case-insensitive comparison, use `TxtCaselessCompare()`.

The `s1MatchLen` and `s2MatchLen` parameters are not as useful for the `TxtCompare` function as they are for the `TxtCaselessCompare` function because `TxtCompare` implements a multi-pass sort algorithm. (See the Compatibility section below for further details.) For example, comparing the string "celery" with the string "Cauliflower" returns a positive value to indicate that "celery" sorts after "Cauliflower," and it returns a match length of 1 to indicate that the second letter determines the sort order ("e" comes after "a"). However, because `TxtCompare` ultimately does a case-sensitive comparison, comparing the string "c" to the string "C" produces a negative result and a match length of 0.

In certain character encodings (such as Shift JIS), one character may be accurately represented as either a single-byte character or a multi-byte character. `TxtCompare` accurately matches a single-byte character with its multi-byte equivalent. For this reason, the values returned in `s1MatchLen` and `s2MatchLen` are not always equal.

You must make sure that the parameters `s1` and `s2` point to the start of a a valid character. That is, they must point to the first byte of a multi-byte character or they must point to a single-byte character; if they don't, results are unpredictable.

Compatibility

Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCompare`. For more information, see Chapter 80, "PalmOSGlue Library."

Prior to Palm OS 4.0, `TxtCompare` and `StrCompare()` only performed one level of comparison and returned as soon as they found two unequal characters. For example, if you compared the string "celery" with the string "Cauliflower," both functions

returned a value indicating that "celery" should appear before "Cauliflower" because they sorted "c" before "C."

In Palm OS 4.0, `StrCompare` calls `TxtCompare`, and `TxtCompare` performs a comparison using up to six comparison tables for sorting with increasing precision. As a result, in Palm OS 4.0 and higher, `TxtCompare` sorts "Cauliflower" before "celery." The `TxtGlueCompare` function uses a two-pass sort on pre-4.0 devices, which will also sort "Cauliflower" before "celery."

Palm OS 4.0 sorting of Shift-JIS characters attempts to duplicate the sorting algorithm described by the JIS standard.

In Palm OS 4.0, the `TxtCompare` function terminates when it finds a null byte in the string. In earlier releases, it terminated only when it reached the ending byte specified by the length parameters.

See Also `StrCompare()`, `TxtFindString()`

TxtConvertEncoding Function

Purpose Convert a text buffer from one character encoding to another.

Declared In `TextMgr.h`

Prototype

```
Err TxtConvertEncoding (Boolean newConversion,
   TxtConvertStateType *ioStateP,
   const Char *srcTextP, UInt16 *ioSrcBytes,
   CharEncodingType srcEncoding, Char *dstTextP,
   UInt16 *ioDstBytes,
   CharEncodingType dstEncoding,
   const Char *substitutionStr,
   UInt16 substitutionLen)
```

Parameters

→ *newConversion*

Set to `true` if this function call is starting a new conversion, or `false` if this function call is a continuation of a previous conversion.

↔ *ioStateP*

If `newConversion` is `false`, this parameter must point to a `TxtConvertStateType` structure containing the same data used for the previous invocation. If `newConversion` is `true` and no subsequent calls are planned, this parameter can be `NULL`.

→ *srcTextP*
: Pointer to the source text buffer. If `newConversion` is `true`, this must point to the start of a text buffer. If `newConversion` is `false`, it may point to a location in the middle of a text buffer. In either case, it must point to an inter-character boundary.

↔ *ioSrcBytes*
: A pointer to the length, in bytes, of the text starting at `srcTextP` that needs to be converted. Upon return, `*ioSrcBytes` contains the number of bytes successfully processed.

→ *srcEncoding*
: The character encoding that the source text uses. This should be one of the Character Encoding Constants.

→ *dstTextP*
: Pointer to the destination text buffer, or `NULL` if you only want to know how large the destination text buffer must be (the minimum size is returned in `*ioDstBytes`). If not `NULL`, `dstTextP` should point to a location where `TxtConvertEncoding()` can begin writing.

→ *ioDstBytes*
: A pointer to the length, in bytes, of `dstTextP`. Upon return, `*ioDstBytes` contains the number of bytes required to represent the source text in the new encoding.

→ *dstEncoding*
: The character encoding to which to convert `srcTextP`. This should be one of the constants documented in "Character Encoding Constants" on page 825. Note that the encoding can be modified, giving you greater control over the conversion process: see "Character Encoding Constant Modifiers" on page 1063.

→ *substitutionStr*
: A string to be substituted for any invalid or inconvertible characters that occur in the source text. This string must already be in the destination encoding. If `NULL`, `TxtGlueConvertEncoding()` immediately returns if it encounters an invalid character.

→ *substitutionLen*
The number of bytes in `substitutionStr`, not including the terminating null byte.

Returns Returns `errNone` upon success or one of the following if an error occurs:

`txtErrConvertOverflow`
The destination buffer is not large enough to contain the converted text.

`txtErrConvertUnderflow`
The end of the source buffer contains a partial character.

`txtErrMalformedText`
An error in the source text encoding has been discovered.

`txtErrNoCharMapping`
The device does not contain a mapping between the source and destination encodings for at least one of the characters in `srcTextP`, and there is no substitution string.

`txtErrUnknownEncoding`
One of the specified encodings is unknown or can't be handled, or the function is being called on a handheld running a version of Palm OS prior to 3.5.

`txtErrUnknownEncodingFallbackCopy`
Either the source or destination encoding is unknown, and the best fit flag was set in the destination encoding. Before returning this error code, `TxtConvertEncoding()` copies anything that is 7-bit ASCII from the source text buffer to the destination text buffer.

Comments This function converts `ioSrcBytes` of text in `srcTextP` from the `srcEncoding` to the `dstEncoding` character encoding and returns the result in `dstTextP`.

`TxtConvertEncoding()` can convert between the following "built-in" encodings and the device encoding:

- UTF-8
- UTF-16, UTF-16BE, UTF-16LE, UCS-2
- UTF-32, UTF-32BE, UTF-32LE, UCS-4[2]

It can also support a variety of encodings that are locale dependent. The only locale for which `TxtConvertEncoding()` supports

additional encodings is simplified Chinese in Palm OS versions that support simplified Chinese. The additional source encodings supported for the simplified Chinese locale are:

- GB2312
- GBK
- ISO2022-CN
- Big5
- Big5+HKSCS
- CP1252
- ISO8859-1
- US ASCII (ISO646)
- GSM

The additional destination encodings supported for the simplified Chinese locale are:

- GB2312
- GBK
- ISO2022-CN
- GSM

If the `charEncodingDstBestFitFlag` is supported (see "Character Encoding Constant Modifiers" on page 1063) but is *not* set in the destination encoding passed to the function, and you specify an unsupported character encoding for either the source or the destination buffer, `TxtConvertEncoding()` returns `txtErrUnknownEncoding`.

You can retrieve the device's encoding using the following function:

```
FtrGet(sysFtrCreator, sysFtrNumEncoding,
  &encoding)
```

If you're converting text that was received from the Internet, the encoding's name is passed along with the text data. Use the

2. UTF-32, UTF-32BE, UTF-32LE, and UCS-4 are only built into handhelds running a version of Palm OS that supports simplified Chinese.

TxtNameToEncoding() function to convert the name to a CharEncodingType value.

The dstTextP buffer must be large enough to hold the result of converting srcTextP to the specified encoding. You can pass NULL for the dstTextP parameter to determine the required length of the buffer before actually doing the conversion; the required length is returned in ioDstBytes.

If the function encounters an inconvertible character in the source text, it puts substitutionStr in the destination buffer in that character's place and continues the conversion. When the conversion is complete, it returns txtErrNoCharMapping to indicate that an error occurred (assuming that no other higher-priority error occurred during the conversion). If substitutionStr is NULL, the function stops the conversion and immediately returns txtErrNoCharMapping. ioSrcBytes is set to the offset of the inconvertible character, dstTextP contains the converted string up to that point, and ioDstBytes contains the size of the converted text. You can examine the character at ioSrcBytes and choose to move past it and continue the conversion. Follow the rules for making repeated calls to TxtConvertEncoding() as described in the next paragraph.

You can make repeated calls to TxtConvertEncoding() in a loop if you only want to convert part of the input buffer at a time. When you make repeated calls to this function, the first call should use true for newConversion, and srcTextP should point to the start of the text buffer. All subsequent calls should use the following values:

newConversion
: false.

ioStateP
: The same data that was returned by the previous invocation.

srcTextP
: The location where this call should begin converting. Typically, this would be the previous srcTextP plus the number of bytes returned in ioSrcBytes.

  If you are skipping over an inconvertible character, srcTextP must point to the character after that location.

↔ *ioSrcBytes*
: The number of bytes that this function call should convert.

`dstTextP`
: A pointer to a location where this function can begin writing the converted string. You might choose to have each function call write to a different destination buffer. To have successive calls write to the same buffer, pass the previous `dstTextP` plus the number of bytes returned in `ioDstBytes` each time.

`ioDstBytes`
: The number of bytes available for output in the `dstTextP` buffer. In other words, the number of bytes remaining.

Encodings Supported by Various ROMs

The locale module is designed to provide support for Unicode, the device encoding, and a set of related or locale-important encodings. The following tables summarize the set of encodings supported in `TxtConvertEncoding()` by various ROMs.

The encoding returned by the Locale Manager in Palm OS Garnet version 5.3SC (or in an earlier release if you use `TxtGlueConvertEncoding()`) for `lmChoiceInboundDefaultVObjectEncoding` is supported as a source encoding, and the encodings returned for `lmChoicePrimarySMSEncoding`, `lmChoiceSecondarySMSEncoding`, `lmChoicePrimaryEmailEncoding`, `lmChoiceSecondaryEmailEncoding`, and `lmChoiceOutboundVObjectEncoding` are supported as destination encodings. The exception to this rule is `charEncodingAscii`, which gets returned by `LmGlueGetLocaleSetting(lmChoicePrimaryEmailEncoding)` on systems running a version of Palm OS prior to 5.3SC. This encoding—`charEncodingAscii`—isn't supported as a destination encoding on Palm OS prior to 5.3SC.

Table 55.1 Source encodings for Latin ROMs version 4.0, 4.1, 5.0, 5.1, 5.2

`charEncodingUCS2`	`charEncodingUTF8`
`charEncodingUTF16LE`	`charEncodingUTF16BE`
`charEncodingPalmLatin`	

Table 55.2 Destination encodings for Latin ROMs version 4.0, 4.1, 5.0, 5.1, 5.2

`charEncodingUCS2`	`charEncodingUTF8`
`charEncodingUTF16LE`	`charEncodingUTF16BE`
`charEncodingPalmLatin`	

Table 55.3 Source encodings for Shift-JIS ROMs version 4.0, 4.1, 5.0, 5.1, 5.2

`charEncodingUCS2`	`charEncodingUTF8`
`charEncodingUTF16LE`	`charEncodingUTF16BE`
`charEncodingPalmSJIS`	

Table 55.4 Destination encodings for Shift-JIS ROMs version 4.0, 4.1, 5.0, 5.1, 5.2

`charEncodingUCS2`	`charEncodingUTF8`
`charEncodingUTF16LE`	`charEncodingUTF16BE`
`charEncodingPalmSJIS`	

Table 55.5 Source encodings for GB ROMs version 5.3SC

`charEncodingUCS2`	`charEncodingUTF8`
`charEncodingUTF16`	`charEncodingUTF16LE`

Table 55.5 Source encodings for GB ROMs version 5.3SC

`charEncodingUTF16BE`	`charEncodingUTF32`
`charEncodingUTF32BE`	`charEncodingUTF32LE`
`charEncodingUCS4`	`charEncodingPalmGB`
`charEncodingGB2312`	`charEncodingGBK`
`charEncodingISO2022CN`	`charEncodingBig5`
`charEncodingBig5_HKSCS`	`charEncodingAscii`
`charEncodingCP1252`	`charEncodingISO8859_1`
`charEncodingGSM`	

Table 55.6 Destination encodings for GB ROMs version 5.3SC

`charEncodingUCS2`	`charEncodingUTF8`
`charEncodingUTF16`	`charEncodingUTF16LE`
`charEncodingUTF16BE`	`charEncodingUTF32`
`charEncodingUTF32BE`	`charEncodingUTF32LE`
`charEncodingUCS4`	`charEncodingPalmGB`
`charEncodingGB2312`	`charEncodingGBK`
`charEncodingISO2022CN`	`charEncodingAscii`
`charEncodingISO8859_1`	`charEncodingGSM`

Compatibility

Implemented only if 4.0 New Feature Set is present. To use this function in code intended to be run on Palm OS 3.5, link with the PalmOSGlue library and call `TxtGlueConvertEncoding()`. For more information, see Chapter 80, "PalmOSGlue Library."

In Palm OS versions that do not support simplified Chinese, the focus of `TxtConvertEncoding()` is to convert between Unicode-encoded text and the device's character encoding. For this reason, `TxtConvertEncoding()` can only handle conversions between the device's encoding and one of UTF-8, UCS-2, UTF-16LE, or UTF-

16BE. If you specify any other character encoding for either the source or the destination buffer, the error code `txtErrUnknownEncoding` is returned.

Prior to Palm OS Garnet version 5.3SC, due to a bug in the implementation of this function the destination buffer must be larger than what is needed in order to accommodate the destination text. If the destination buffer is exactly the right size, an error is returned. Accordingly, if you call `TxtConvertEncoding()` and pass `NULL` for *`dstTextP`* in order to determine the needed buffer size, add one to the returned buffer size when you allocate the buffer.

TxtEncodingName Function

Purpose Obtain a character encoding's name.

Declared In `TextMgr.h`

Prototype

```
const Char *TxtEncodingName
    (CharEncodingType inEncoding)
```

Parameters → *`inEncoding`*
One of the values from `CharEncodingType`, indicating a character encoding.

Returns A constant string containing the name of the encoding. The possible return values are defined in `PalmLocale.h`.

Comments Use this function to obtain the official name of the character encoding, suitable to pass to an Internet application or any other application that requires the character encoding's name to be passed along with the data.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueEncodingName`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `TxtNameToEncoding()`

TxtFindString Function

Purpose Perform a case-insensitive search for a string in another string.

Declared In `TextMgr.h`

Prototype

```
Boolean TxtFindString (const Char *inSourceStr,
   const Char *inTargetStr, UInt32 *outPos,
   UInt16 *outLength)
```

Parameters

→ *inSourceStr*
Pointer to the string to be searched.

→ *inTargetStr*
Prepared version of the string to be found. This string should either be passed directly from the `strToFind` field in the `sysAppLaunchCmdFind` launch code's parameter block or it should be prepared using the PalmOSGlue function `TxtGluePrepFindString()`.

← *outPos*
Pointer to the offset of the match in `inSourceStr`.

← *outLength*
Pointer to the length in bytes of the matching text.

Returns Returns `true` if the function finds `inTargetStr` within `inSourceStr`; `false` otherwise.

If found, the values pointed to by the `outPos` and `outLength` parameters are set to the starting offset and the length of the matching text. If not found, the values pointed to by `outPos` and `outLength` are set to 0.

The search that `TxtFindString` performs is locale-dependent. On most ROMs with Latin-based encodings, `TxtFindString` returns `true` only if the string is at the beginning of a word. On Shift-JIS encoded ROMs, `TxtFindString` returns `true` if the string is located anywhere in the word.

Comments Use this function instead of `FindStrInStr()` to support the global system find facility. This function contains an extra parameter, `outLength`, to specify the length of the text that matched. Pass this value to `FindSaveMatch()` in the `appCustom` parameter. Then when your application receives `sysAppLaunchCmdGoTo`, the `matchCustom` field contains the

length of the matching text. You use the length of matching text to highlight the match within the selected record.

You must make sure that the parameters `inSourceStr` and `inTargetStr` point to the start of a valid character. That is, they must point to the first byte of a multi-byte character, or they must point to a single-byte character; if they don't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueFindString()`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also TxtCaselessCompare()

TxtGetChar Function

Purpose Retrieve the character starting at the specified offset within a text buffer.

Declared In `TextMgr.h`

Prototype

```
WChar TxtGetChar (const Char *inText,
    UInt32 inOffset)
```

Parameters

→ *`inText`*
Pointer to the text buffer to be searched.

→ *`inOffset`*
A valid offset into the buffer `inText`. This offset must point to an inter-character boundary.

Returns Returns the character at `inOffset` in `inText`.

Comments You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS,

link with the PalmOSGlue library and call `TxtGlueGetChar`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also TxtGetNextChar(), TxtSetNextChar()

TxtGetNextChar Function

Purpose Retrieve the character starting at the specified offset within a text buffer.

Declared In `TextMgr.h`

Prototype

```
UInt16 TxtGetNextChar (const Char *inText,
    UInt32 inOffset, WChar *outChar)
```

Parameters

→ *inText*
Pointer to the text buffer to be searched.

→ *inOffset*
A valid offset into the buffer `inText`. This offset must point to an inter-character boundary.

← *outChar*
The character at `inOffset` in `inText`. Pass `NULL` for this parameter if you don't need the character returned.

Returns Returns the size in bytes of the character at `inOffset`. If `outChar` is not `NULL` upon entry, it points to the character at `inOffset` upon return.

Comments You can use this function to iterate through a text buffer character-by-character in this way:

```
UInt16 i = 0;
WChar ch;
while (i < bufferLength) {
    i += TxtGetNextChar(buffer, i, &ch);
    //do something with ch.
}
```

You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueGetNextChar`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also TxtGetChar(), TxtGetPreviousChar(), TxtSetNextChar()

TxtGetPreviousChar Function

Purpose Retrieve the character before the specified offset within a text buffer.

Declared In `TextMgr.h`

Prototype

```
UInt16 TxtGetPreviousChar (const Char *inText,
    UInt32 inOffset, WChar *outChar)
```

Parameters

→ *inText*
Pointer to the text buffer to be searched.

→ *inOffset*
A valid offset into the buffer `inText`. This offset must point to an inter-character boundary.

← *outChar*
The character immediately preceding `inOffset` in `inText`. Pass `NULL` for this parameter if you don't need the character returned.

Returns Returns the size in bytes of the character preceding `inOffset` in `inText`. If `outChar` is not `NULL` upon entry, then it points to the character preceding `inOffset` upon return. Returns 0 if `inOffset` is at the start of the buffer (that is, `inOffset` is 0).

Comments You can use this function to iterate through a text buffer character-by-character in this way:

```
WChar ch;
/* Find the start of the character containing
the last byte. */
TxtCharBounds (buffer, bufferLength - 1,
&start, &end);
i = start;
while (i > 0) {
    i -= TxtGetPreviousChar(buffer, i, &ch);
    //do something with ch.
}
```

This function is often slower to use than `TxtGetNextChar()` because it must determine the appropriate character boundaries if the byte immediately before the offset is valid in more than one location (start, middle, or end) of a multi-byte character. To do this, it must work backwards toward the beginning of the string until it finds an unambiguous byte.

You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueGetPreviousChar`. For more information, see Chapter 80, "PalmOSGlue Library."

TxtGetTruncationOffset Function

Purpose Return the appropriate byte position for truncating a text buffer such that it is at most a specified number of bytes long.

Declared In `TextMgr.h`

Prototype

```
UInt32 TxtGetTruncationOffset
    (const Char *inText,
    UInt32 inOffset)
```

Parameters

→ *inText*
Pointer to a text buffer.

→ *inOffset*
An offset into the buffer `inText`.

Returns Returns the appropriate byte offset for truncating `inText` at a valid inter-character boundary. The return value may be less than or equal to `inOffset`.

Comments You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueGetTruncationOffset`. For more information, see Chapter 80, "PalmOSGlue Library."

TxtGetWordWrapOffset Function

Purpose Locate an appropriate place for a line break in a text buffer.

Declared In `TextMgr.h`

Prototype

```
UInt32 TxtGetWordWrapOffset (const Char *iTextP,
    UInt32 iOffset)
```

Parameters

→ *`iTextP`*
Pointer to a text buffer.

→ *`iOffset`*
Pointer to the offset where the search should begin. The search is performed backward starting from this offset.

Returns Returns the offset of a character that can begin on a new line (typically, the beginning of the word that contains `iOffset` or last word before `iOffset`). If an appropriate break could not be found, returns `iOffset`.

Comments The `FntWordWrap()` and `FntWordWrapReverseNLines()` functions call `TxtGetWordWrapOffset` to locate an appropriate place to break the text. The returned offset points to the character that should begin the next line.

This function starts at `iOffset` and works backward until it finds a character that typically occurs between words (for example, white space or punctuation). Then it moves forward until it locates the character that begins a word (typically, a letter or number). Note that this function may return an offset value that is greater than the one passed in if the offset passed in occurs immediately before white space or in the middle of white space.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TxtWordBounds()`

TxtMaxEncoding Function

Purpose Return the higher of two encodings.

Declared In `TextMgr.h`

Prototype

```
CharEncodingType TxtMaxEncoding
    (CharEncodingType a, CharEncodingType b)
```

Parameters

→ *a*
A character encoding to compare.

→ *b*
Another character encoding to compare.

Returns Returns the higher of `a` or `b`. One character encoding is higher than another if it is more specific. For example code page 1252 is "higher" than ISO 8859-1 because it represents more characters than ISO 8859-1.

Comments This function is used by `TxtStrEncoding()` to determine the encoding required for a string.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueMaxEncoding`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `TxtCharEncoding()`, `CharEncodingType`

TxtNameToEncoding Function

Purpose Return an encoding's constant given its name.

Declared In `TextMgr.h`

Prototype

```
CharEncodingType TxtNameToEncoding
    (const Char *iEncodingName)
```

Parameters

→ *iEncodingName*
One of the string constants containing the official name of an encoding. See `PalmLocale.h` for the complete set of string constants.

Returns Returns one of the Character Encoding Constants. Returns `charEncodingUnknown` if the specified encoding could not be found.

Comments Use this function to convert a character encoding's name as received from an Internet application into the character encoding constant that some text manager functions require.

This function properly converts aliases for a character encoding. For example, passing the strings "us-ascii", "ASCII", "cp367", and "IBM367" all return `charEncodingAscii`.

The known character encodings are device-dependent. For example, a device with the Shift-JIS encoding will not know all of the aliases for Latin character encodings; however, it will know all of the aliases for Shift-JIS.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TxtEncodingName()`

TxtNextCharSize Macro

Purpose Returns the size of the character starting at the specified offset within a text buffer.

Declared In `TextMgr.h`

Prototype

```
#define TxtNextCharSize (inText, inOffset)
```

Parameters

→ *inText*
Pointer to the text buffer to be searched.

→ *inOffset*
A valid offset into the buffer `inText`. This offset must point to an inter-character boundary.

Returns Returns (as a `UInt16`) the size in bytes of the character at `inOffset`.

Comments You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

Compatibility Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueNextCharSize`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `TxtGetNextChar()`

TxtParamString Function

Purpose Replace substrings within a string with the specified values.

Declared In `TextMgr.h`

Prototype

```
Char *TxtParamString (const Char *inTemplate,
   const Char *param0, const Char *param1,
   const Char *param2, const Char *param3)
```

Parameters

→ *inTemplate*
The string containing the substrings to replace.

→ *param0*
String to replace ^0 with or `NULL`.

→ *param1*
String to replace ^1 with or `NULL`.

→ *param2*
String to replace ^2 with or `NULL`.

→ *param3*
String to replace ^3 with or `NULL`.

Returns Returns a pointer to a locked relocatable chunk in the dynamic heap that contains the appropriate substitutions.

Comments This function searches `inTemplate` for occurrences of the sequences ^0, ^1, ^2, and ^3. When it finds these, it replaces them with the corresponding string passed to this function. Multiple instances of each sequence will be replaced.

The replacement strings can also contain the substitution strings, provided they refer to a later parameter. That is, the `param0` string contain have references to ^1, ^2, and ^3, the `param1` string can have references to ^2 and ^3, and the `param2` string can have references to ^3. Any other occurrences of the substitution strings in the replacement strings are ignored. For example, if `param3` is the string "^0", any occurrences of ^3 in `inTemplate` are replaced with the string "^0".

You must make sure that the parameter `inTemplate` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

`TxtParamString` allocates space for the returned string in the dynamic heap through a call to `MemHandleNew`, and then returns the result of calling `MemHandleLock` with this handle. Your code is responsible for freeing this memory when it is no longer needed.

Compatibility Implemented if 3.5 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueParamString`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `TxtReplaceStr()`, `FrmCustomAlert()`

TxtPreviousCharSize Macro

Purpose Returns the size of the character before the specified offset within a text buffer.

Declared In `TextMgr.h`

Prototype

```
#define TxtPreviousCharSize (inText, inOffset)
```

Parameters

→ *inText*
Pointer to the text buffer.

→ *inOffset*
A valid offset into the buffer `inText`. This offset must point to an inter-character boundary.

Returns Returns (as a `UInt16`) the size in bytes of the character preceding `inOffset` in `inText`. Returns 0 if `inOffset` is at the start of the buffer (that is, `inOffset` is 0).

Comments You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

This macro is often slower to use than `TxtNextCharSize()` because it must determine the appropriate character boundaries if the byte immediately before the offset is valid in more than one location (start, middle, or end) of a multi-byte character. To do this, it must work backwards toward the beginning of the string until it finds an unambiguous byte.

Compatibility Valid only if International Feature Set is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGluePreviousCharSize`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also `TxtGetPreviousChar()`

TxtReplaceStr Function

Purpose Replace a substring of a given format with another string.

Declared In `TextMgr.h`

Prototype

```
UInt16 TxtReplaceStr (Char *ioStr,
   UInt16 inMaxLen, const Char *inParamStr,
   UInt16 inParamNum)
```

Parameters

↔ *ioStr*
The string in which to perform the replacing.

→ *inMaxLen*
The maximum length in bytes that `ioStr` can become.

→ *inParamStr*
The string that ^*inParamNum* should be replaced with. If `NULL`, no changes are made.

→ *inParamNum*
A single-digit number (0 to 9).

Returns Returns the number of occurrences found and replaced.

Returns a fatal error message if `inParamNum` is greater than 9.

Comments This function searches `ioStr` for occurrences of the string ^*inParamNum*, where *inParamNum* is any digit from 0 to 9. When it finds the string, it replaces it with `inParamStr`. Multiple instances will be replaced as long as the resulting string doesn't contain more than `inMaxLen` bytes, not counting the terminating null.

You can set the `inParamStr` parameter to `NULL` to determine the required length of `ioStr` before actually doing the replacing. `TxtReplaceStr` returns the number of occurrences it finds of ^*inParamNum*. Multiply this value by the length of the

`inParamStr` you intend to use to determine the appropriate length of `ioStr`.

You must make sure that the parameter `ioStr` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueReplaceStr`. For more information, see Chapter 80, "PalmOSGlue Library."

TxtSetNextChar Function

Purpose Set a character within a text buffer.

Declared In `TextMgr.h`

Prototype

```
UInt16 TxtSetNextChar (Char *ioText,
    UInt32 inOffset, WChar inChar)
```

Parameters

↔ *ioText*
Pointer to a text buffer.

→ *inOffset*
A valid offset into the buffer `inText`. This offset must point to an inter-character boundary.

→ *inChar*
The character to replace the character at `inOffset` with. Must not be a virtual character.

Returns Returns the size of `inChar`.

Comments This function replaces the character in `ioText` at the location `inOffset` with the character `inChar`. Note that there must be enough space at `inOffset` to write the character.

You can use `TxtCharSize()` to determine the size of `inChar`.

You must make sure that the parameter `ioText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueSetNextChar`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also TxtGetNextChar()

TxtStrEncoding Function

Purpose Return the encoding required to represent a string.

Declared In `TextMgr.h`

Prototype

```
CharEncodingType TxtStrEncoding
    (const Char *inStr)
```

Parameters → *inStr*
A string.

Returns A `CharEncodingType` value that indicates the encoding required to represent `inChar`. If any character in the string isn't recognizable, then `charEncodingUnknown` is returned.

Comments The encoding for the string is the maximum encoding of any character in that string. For example, if a two-character string contains a blank space and a ü, the appropriate encoding is `charEncodingISO8859_1`. The blank space's minimum encoding is ASCII. The minimum encoding for the ü is ISO 8859-1. The maximum of these two encodings is ISO 8859-1.

Use this function for informational purposes only. Your code should not assume that the character encoding returned by this function is the Palm OS system's character encoding. (Instead use `FtrGet` as shown in the `TxtCharXAttr()` function description.)

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueStrEncoding`. For more information, see Chapter 80, "PalmOSGlue Library."

Palm OS only supports a single character encoding at a time. Because of this, the value returned from this function is always logically equal to or less than the encoding used on the current system. That is, you'll only receive a return value of

`charEncodingISO8859_1` if you're running on a Latin-based system.

See Also `TxtCharEncoding()`, `TxtMaxEncoding()`

TxtTransliterate Function

Purpose Converts the specified number of bytes in a text buffer using the specified operation.

Declared In `TextMgr.h`

Prototype

```
Err TxtTransliterate (const Char *inSrcText,
   UInt16 inSrcLength, Char *outDstText,
   UInt16 *ioDstLength, TranslitOpType inOp)
```

Parameters

→ *inSrcText*
: Pointer to a text buffer.

→ *inSrcLength*
: The length in bytes of `inSrcText`.

← *outDstText*
: The output buffer containing the converted characters.

↔ *ioDstLength*
: Upon entry, the maximum length of `outDstText`. Upon return, the actual length of `outDstText`.

→ *inOp*
: A 16-bit unsigned value that specifies which transliteration operation is to be performed. See `TranslitOpType` for the possible values for this field.

Returns one of the following values:

`errNone`
: Success

`txtErrUknownTranslitOp`
: `inOp`'s value is not recognized

`txtErrTranslitOverrun`
: `inSrcText` and `outDstText` point to the same memory location and the operation has caused the function to overwrite unprocessed data in the input buffer.

`txtErrTranslitOverflow`
: `outDstText` is not large enough to contain the converted string.

`txtErrTranslitUnderflow`
: The end of the source buffer contains a partial character.

Comments

`inSrcText` and `outDstText` may point to the same location if you want to perform the operation in place. However, you should be careful that the space required for `outDstText` is not larger than `inSrcText` so that you don't generate a `txtErrTranslitOverrun` error.

For example, suppose on a Shift JIS encoded system, you want to convert a series of single-byte Japanese Katakana symbols to double-byte Katakana symbols. You cannot perform this operation in place because it replaces a single-byte character with a multi-byte character. When the first converted character is written to the buffer, it overwrites the second input character. Thus, a text overrun has occurred.

You can ensure that you have enough space for the output by OR-ing your chosen operation with `translitOpPreprocess`. For example, to convert a string to uppercase letters, do the following:

```
outSize = buf2Len;
error = TxtTransliterate(buf1, buf1len, &buf2,
&outSize,
translitOpUpperCase|translitOpPreprocess);
if (outSize > buf2len)
    /* allocate more memory for buf2 */
error = TxtTransliterate(buf1, buf1Len, &buf2,
&outSize, translitOpUpperCase);
```

You must make sure that the parameter `inSrcText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

Compatibility

Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueTransliterate`. For more information, see Chapter 80, "PalmOSGlue Library."

In Palm OS 4.0, the `TxtTransliterate` function terminates when it finds a null byte in the source string. In earlier releases, it terminated only when it reached the ending byte specified by the length parameter.

TxtWordBounds Function

Purpose Find the boundaries of a word of text that contains the character starting at the specified offset.

Declared In `TextMgr.h`

Prototype

```
Boolean TxtWordBounds (const Char *inText,
   UInt32 inLength, UInt32 inOffset,
   UInt32 *outStart, UInt32 *outEnd)
```

Parameters

→ *inText*
Pointer to a text buffer.

→ *inLength*
The length in bytes of the text pointed to by `inText`.

→ *inOffset*
A valid offset into the text buffer `inText`. This offset must point to the beginning of a character.

← *outStart*
The starting offset of the text word.

← *outEnd*
The ending offset of the text word.

Returns Returns `true` if a word is found. Returns `false` if the word doesn't exist or is punctuation or whitespace.

Comments Assuming the ASCII encoding, if the text buffer contains the string "Hi! How are you?" and you pass 5 as the offset, `TxtWordBounds` returns the start and end of the word containing the character at offset 5, which is the character "o". Thus, `outStart` and `outEnd` would point to the start and end of the word "How".

You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

Compatibility Implemented only if International Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueWordBounds`. For more information, see Chapter 80, "PalmOSGlue Library."

In Palm OS 4.0, the `TxtWordBounds` function terminates when it finds a null byte in the string. In earlier releases, it terminated only when it reached the ending byte specified by the length parameter.

See Also `TxtCharBounds()`, `TxtCharIsDelim()`, `TxtGetWordWrapOffset()`

56

Text Services Manager

This chapter provides information about the Text Services Manager API as declared in `TextServicesMgr.h`. The Text Services Manager provides the caller with an API for interacting with various text services, including front-end processors (FEPs), which are sometimes known as input methods. This chapter discusses the following topics:

- Text Services Manager Data Structures
- Text Services Manager Functions

Text Services Manager Data Structures

TsmFepModeType Typedef

Purpose The `TsmFepModeType` type specifies the input modes used by the functions `TsmGetFepMode()` and `TsmSetFepMode()`.

Prototype

```
typedef UInt16 TsmFepModeType;
```

Comments `TsmFepModeType` can be set to one of the defined modes listed in the following table.

Constant	Value	Description
`tsmFepModeDefault`	0	The default input mode for the FEP. For example, with the Japanese FEP, the default mode is Hiragana.

Constant	Value	Description
`tsmFepModeOff`	1	Indicates that there is no active FEP input mode (the FEP is off).
`tsmFepModeCustom`	128	A custom FEP input mode. You can have more than one custom mode; the starting value is 128. Katagana is an example of a custom input mode for the Japanese FEP.

Text Services Manager Functions

TsmGetFepMode Function

Purpose Return the current input mode for the active front-end processor (FEP).

Declared In `TextServicesMgr.h`

Prototype

```
TsmFepModeType TsmGetFepMode (void *nullParam)
```

Parameters

→ *`nullParam`*
An unused status pointer that must be set to `NULL`.

Returns If there is an active FEP, returns the current mode for the active FEP. If there is no active FEP, returns `tsmFepModeOff`.

Comments The most common use for this function is to save the current FEP mode. You could then call `TsmSetFepMode()` to set the current mode to "off" and again to restore the saved mode once the application has finished using a special text field.

Compatibility Implemented only if 3.5 New Feature Set is present. In Palm OS 3.5, the `nullParam` parameter takes a non-`NULL` value, allowing the caller to maintain its own status record. In Palm OS 4.0, this parameter is unused and must be set to `NULL`. Any other value generates a non-fatal alert.

See Also `TsmSetFepMode()`

TsmSetFepMode Function

Purpose Set the input mode for the active front-end processor (FEP) to be the mode defined by the parameter `inNewMode`.

Declared In `TextServicesMgr.h`

Prototype

```
TsmFepModeType TsmSetFepMode (void *nullParam,
    TsmFepModeType inNewMode)
```

Parameters

→ *`nullParam`*
An unused status pointer that must be set to `NULL`.

→ *`inNewMode`*
The new FEP input mode.

Returns Returns the previous input mode. If there is no active FEP, returns `tsmFepModeOff`.

Comments The most common use for this function is to set the FEP mode to "off" while the application is using a special text field, and then to restore the previous mode. See `TsmGetFepMode()` for more information on saving and restoring the FEP mode.

One common reason for explicitly disabling the FEP in code is when a text field will only contain 7-bit ASCII (numeric fields automatically turn off the FEP). For example, if the application has a password field and the contents of that field will always be 7-bit ASCII, the application should turn off the FEP to help prevent the user from entering invalid characters into the field.

Another common case occurs when the application has a numeric field, but cannot just rely on the numeric field attribute. For example, if you want the user to be able to enter the minus ("-") sign, you cannot use a numeric field because the field code prevents the user from entering this character: its not a digit or a period. In this case, you should make it a regular field and have the application screen the characters. The application should disable the FEP when such a pseudo-numeric field is active.

IMPORTANT: A mode change is currently enqueued as a keyDown event so that the field and FEP can remain properly synchronized and so that the mode change doesn't affect any pending keyDown events. The mode change does not happen until the enqueued keyDown event is passed to `FrmHandleEvent`; if you call `TsmGetFepMode()` immediately after calling `TsmSetFepMode()`, you won't see a mode change.

There are also some current limitations with changing the mode: there must be an active form; and if there is an active field in the form, it must not be a numeric field.

Compatibility Implemented only if 3.5 New Feature Set is present. In Palm OS 3.5, the `nullParam` parameter takes a non-`NULL` value, allowing the caller to maintain its own status record. In Palm OS 4.0, this parameter is unused and must be set to `NULL`. Any other value generates a non-fatal alert.

See Also TsmGetFepMode(), EvtEnqueueKey(), FrmHandleEvent()

57

Time Manager

This chapter provides reference material for the time manager.

- Time Manager Data Structures
- Time Manager Functions

The header file `DateTime.h` declares the API that this chapter describes. For more information on the time manager, see the section "Time" in the *Palm OS Programmer's Companion*, vol. I.

Time Manager Data Structures

The time manager uses these structures to store information.

DateFormatType Enum

Purpose The `DateFormatType` enum specifies the different display formats for date values.

Prototype

```
typedef enum {
   dfMDYWithSlashes,
   dfDMYWithSlashes,
   dfDMYWithDots,
   dfDMYWithDashes,
   dfYMDWithSlashes,
   dfYMDWithDots,
   dfYMDWithDashes,
   dfMDYLongWithComma,
   dfDMYLong,
   dfDMYLongWithDot,
   dfDMYLongNoDay,
   dfDMYLongWithComma,
   dfYMDLongWithDot,
```

```
   dfYMDLongWithSpace,
   dfMYMed,
   dfMYMedNoPost,
   dfMDYWithDashes
} DateFormatType;
```

Constants

`dfMDYWithSlashes`
: The month, day, and year numbers separated by slashes. For example, `12/31/95`.

  This is considered a short format.

`dfDMYWithSlashes`
: The day, month, and year numbers separated by slashes. For example, `31/12/95`.

  This is considered a short format.

`dfDMYWithDots`
: The day, month, and year numbers separated by dots. For example, `31.12.95`.

  This is considered a short format.

`dfDMYWithDashes`
: The day, month, and year numbers separated by dashes. For example, `31-12-95`.

  This is considered a short format.

`dfYMDWithSlashes`
: The year, month, and day numbers separated by slashes. For example, `95/12/31`.

  This is considered a short format.

`dfYMDWithDots`
: The year, month, and day numbers separated by dots. For example, `95.12.31`.

  This is considered a short format.

`dfYMDWithDashes`
: The year, month, and day numbers separated by dashes. For example, `95-12-31`.

  This is considered a short format.

`dfMDYLongWithComma`
: The month, day, and year in long format, with a comma. For example, `Dec 31, 1995.`

  This is considered a long format.

`dfDMYLong`
: The month, day, and year in long format. For example, `31 Dec 1995.`

  This is considered a long format.

`dfDMYLongWithDot`
: The month, day, and year in long format, with a dot. For example, `31. Dec 1995.`

  This is considered a long format.

`dfDMYLongNoDay`
: The month and year in long format. For example, `Dec 1995.`

  This is considered a long format.

`dfDMYLongWithComma`
: The day, month, and year in long format, with a comma. For example, `31 Dec, 1995.`

  This is considered a long format.

`dfYMDLongWithDot`
: The year, month, and day in long format with dot separators. For example, `1995.12.31.`

  This is considered a long format.

`dfYMDLongWithSpace`
: The year, month, and day in long format with space separators. For example, `1995 Dec 31.`

  This is considered a long format.

`dfMYMed`
: The month in long format with the two-digit year, preceded by an apostrophe. For example, `Dec '95.`

  This is considered a medium format.

`dfMYMedNoPost`
: The month in long format with the two-digit year. For example, `Dec 95`.

  This is considered a medium format.

`dfMDYWithDashes`
: The month, day, and year numbers separated by dashes. For example, `12-31-95.`

  This is considered a short format.

Compatibility The `dfMDYWithDashes` constant is defined in Palm OS® 4.0 and higher.

DateTimeType Struct

Purpose The `DateTimeType` structure represents a date and time value.

Prototype

```
typedef struct{
  Int16 second;
  Int16 minute;
  Int16 hour;
  Int16 day;
  Int16 month;
  Int16 year;
  Int16 weekDay;
} DateTimeType
typedef DateTimeType *DateTimePtr;
```

Fields

`second`
: The number of seconds. This is a value between 0 and 59.

`minute`
: The number of minutes. This is a value between 0 and 59.

`hour`
: The number of hours. This is a value between 0 and 23.

`day`
: The day number. This is a value between 1 and 31.

`month`
: The month number. This is a value between 1 and 12.

`year`
: The year number.

`weekDay`
The day number. This represents the number of days since Sunday and is thus a value between 0 and 6.

DateType Struct

Purpose The `DateType` structure represents a date value.

Prototype

```
typedef struct{
  UInt16 year :7;
  UInt16 month:4;
  UInt16 day:5;
} DateType;
typedef DateType *DatePtr;
```

Fields

`year`
The number of years since 1904.

Note that this is the format used on Macintosh computers.

`month`
The month number. This is a value between 1 and 12.

`day`
The day number. This is a value between 1 and 31.

DaylightSavingsTypes Enum

Purpose The `DaylightSavingsTypes` enum specifies the different forms of daylight savings times that you can specify for date and time values.

Note that the table uses "DST" to represent daylight savings time.

Prototype

```
typedef enum {
   dsNone,
   dsUSA,
   dsAustralia,
   dsWesternEuropean,
   dsMiddleEuropean,
   dsEasternEuropean,
```

```
   dsGreatBritain,
   dsRumania,
   dsTurkey,
   dsAustraliaShifted
} DaylightSavingsTypes;
```

Constants

`dsNone`
No DST (daylight savings time)

`dsUSA`
U.S.A. DST

`dsAustralia`
Australian DST

`dsWesternEuropean`
Western European DST

`dsMiddleEuropean`
Middle European DST

`dsEasternEuropean`
Eastern European DST

`dsGreatBritain`
Great Britain and Eire DST

`dsRumania`
Rumanian DST

`dsTurkey`
Turkish DST

`dsAustraliaShifted`
Australian DST, with the 1986 shift

Compatibility

If 4.0 New Feature Set is present, this data type is obsolete. In versions 4.0 and higher, Palm OS represents daylight savings time as an integer value that gives the number of minutes to add to the current time for daylight savings time.

DayOfMonthType Enum

Purpose

The `DayOfMonth` enum specifies the different day-of-the-week numeric values that are returned by the `DayOfMonth()` function. These values are used to represent repeating appointments that

occur on specific days of the month; for example, the first Friday or the third Tuesday of each month.

Prototype

```
typedef enum {
   dom1stSun, dom1stMon, dom1stTue, dom1stWen,
   dom1stThu, dom1stFri, dom1stSat,
   dom2ndSun, dom2ndMon, dom2ndTue, dom2ndWen,
   dom2ndThu, dom2ndFri, dom2ndSat,
   dom3rdSun, dom3rdMon, dom3rdTue, dom3rdWen,
   dom3rdThu, dom3rdFri, dom3rdSat,
   dom4thSun, dom4thMon, dom4thTue, dom4thWen,
   dom4thThu, dom4thFri, dom4thSat,
   domLastSun, domLastMon, domLastTue,
   domLastWen, domLastThu, domLastFri,
   domLastSat
} DayOfWeekType;
```

Constants

`dom1stSun`
: The first Sunday of the month.

`dom1stMon`
: The first Monday of the month.

`dom1stTue`
: The first Tuesday of the month.

`dom1stWen`
: The first Wednesday of the month.

`dom1stThu`
: The first Thursday of the month.

`dom1stFri`
: The first Friday of the month.

`dom1stSat`
: The first Saturday of the month.

`dom2ndSun`
: The second Sunday of the month.

`dom2ndMon`
: The second Monday of the month.

`dom2ndTue`
: The second Tuesday of the month.

`dom2ndWen`
: The second Wednesday of the month.

`dom2ndThu`
: The second Thursday of the month.

`dom2ndFri`
: The second Friday of the month.

`dom2ndSat`
: The second Saturday of the month.

`dom3rdSun`
: The third Sunday of the month.

`dom3rdMon`
: The third Monday of the month.

`dom3rdTue`
: The third Tuesday of the month.

`dom3rdWen`
: The third Wednesday of the month.

`dom3rdThu`
: The third Thursday of the month.

`dom3rdFri`
: The third Friday of the month.

`dom3rdSat`
: The third Saturday of the month.

`dom4thSun`
: The fourth Sunday of the month.

`dom4thMon`
: The fourth Monday of the month.

`dom4thTue`
: The fourth Tuesday of the month.

`dom4thWen`
: The fourth Wednesday of the month.

`dom4thThu`
: The fourth Thursday of the month.

`dom4thFri`
: The fourth Friday of the month.

`dom4thSat`
: The fourth Saturday of the month.

`domLastSun`
The last Sunday of the month.

`domLastMon`
The last Monday of the month.

`domLastTue`
The last Tuesday of the month.

`domLastWen`
The last Wednesday of the month.

`domLastThu`
The last Thursday of the month.

`domLastFri`
The last Friday of the month.

`domLastSat`
The last Saturday of the month.

Compatibility On Palm OS versions earlier than 4.0, this type was named `DayOfWeekType`.

TimeFormatType Enum

Purpose The `TimeFormatType` enum specifies the different display formats for time values.

Prototype

```
typedef enum {
   tfColon,
   tfColonAMPM,
   tfColon24h,
   tfDot,
   tfDotAMPM,
   tfDot24h,
   tfHoursAMPM,
   tfHours24h,
   tfComma24h,
} TimeFormatType;
typedef TimeFormatType *TimeFormatPtr;
```

Constants

`tfColon`
The hour and minutes separated by a colon character. For example, `1:00`.

`tfColonAMPM`
: The hour and minutes separated by a colon and followed by an AM/PM indication. For example, `1:00 pm`.

`tfColon24h`
: The 24-hour time with the hour and minutes separated by a colon character. For example, `13:00`.

`tfDot`
: The hour and minutes separated by a dot character. For example, `1.00`.

`tfDotAMPM`
: The hour and minutes separated by a period and followed by an AM/PM indication. For example, `1.00 pm`.

`tfDot24h`
: The 24-hour time with the hour and minutes separated by a dot character. For example, `13.00`.

`tfHoursAMPM`
: The hour value followed by an AM/PM indication. For example, `1 pm`.

`tfHours24h`
: The 24-hour value, followed by an AM/PM indication. For example, `13`.

`tfComma24h`
: The 24-hour time with the hour and minutes separated by a comma character. For example, `13,00`.

TimeType Struct

Purpose The `TimeType` structure represents a time value.

Prototype

```
typedef struct {
  UInt8   hours;
  UInt8   minutes;
} TimeType;
typedef TimeType *TimePtr;
```

Fields

`hours`
: The number of hours. This is a value between 0 and 23.

`minutes`
: The number of minutes. This is value between 0 and 59.

Time Manager Constants

The following table shows the constants that represent the maximum lengths of strings returned by the date and time formatting routines DateToAscii(), DateToDOWDMFormat(), and TimeToAscii().

Constant	Value	Description
dateStringLength	9	Maximum length of the string returned by DateToAscii for short date formats.
longDateStrLength	15	Maximum length of the string returned by DateToAscii for medium and long date formats.
timeStringLength	9	Maximum length of the string returned by TimeToAscii.
dowDateStringLength	19	Maximum length of the string returned by DateToDOWDMFormat for short date formats.
dowLongDateStrLength	25	Maximum length of the string returned by DateToDOWDMFormat for both medium and long date formats.

Time Manager Functions

DateAdjust Function

Purpose Return a new date +/- the days adjustment.

Declared In DateTime.h

Prototype

```
void DateAdjust (DateType *dateP,
    Int32 adjustment)
```

Parameters ↔ *dateP*

A pointer to a DateType structure with the date to be adjusted.

→ *adjustment*
The number of days by which to adjust the date.

Returns Returns nothing. Upon return, `dateP` contains the adjusted date.

Comments This function advances the date and manages month and year wrapping conditions.

See Also `TimAdjust()`

DateDaysToDate Function

Purpose Converts a date specified as the number of days since January 1, 1904 to a `DateType` structure.

Declared In `DateTime.h`

Prototype

```
void DateDaysToDate (UInt32 days,
   DateType *dateP)
```

Parameters → *days*
The number of days since 1/1/1904.

↔ *dateP*
A pointer to a `DateType` structure that is updated with the computed date values.

Returns Returns nothing. Upon return, the date information is returned in the structure referenced by the `date` parameter.

See Also DateSecondsToDate(), DateToDays()

DateSecondsToDate Function

Purpose Converts a date specified as the number of seconds since January 1, 1904 to a `DateType` structure.

Declared In `DateTime.h`

Prototype

```
void DateSecondsToDate (UInt32 seconds,
   DateType *dateP)
```

Parameters → *seconds*
The number of seconds since 1/1/1904.

← *dateP*
A pointer to a `DateType` structure that is updated with the computed date values.

Returns Returns nothing. The structure referenced by the `date` parameter is updated with the date information.

See Also DateDaysToDate(), DateToDays()

DateTemplateToAscii Function

Purpose Convert the specified date values into a string that is formatted according to a formatting template specification.

Declared In `DateTime.h`

Prototype

```
UInt16 DateTemplateToAscii (const Char *templateP,
   UInt8 months, UInt8 days, UInt16 years,
   Char *stringP, Int16 stringLen)
```

Parameters

→ *templateP*
A pointer to the template string used to format the date.

See the Comments section below for details on how to specify date formatting in this template string.

→ *months*
The month number, which must be a value between `1` and `12`.

→ *days*
The day number, which must be a value between `1` and `31`.

→ *years*
The four-digit year number. For example, `1995`.

← *stringP*
A pointer to a string that is updated with the result.

If `stringP` is `NULL`, this function does not write an output string; however, it does return the length required for the output string.

If `stringP` is not `NULL`, this function writes the formatted string to `stringP`, writing up to `stringSize` bytes into `stringP`.

→ *stringLen*
The size of the `stringP` buffer.

Returns The length of the formatted string, without the terminating null byte.

The `DateTemplateToAscii` returns the required length of the formatted string even if the `stringP` parameter is `NULL`; this allows you to determine the buffer size at runtime.

Comments This function is intended as a replacement for the `DateToAscii()` and `DateToDOWDMFormat()` functions.

This function uses the formatting template referenced by `templateP` to create a formatted string from the date values that you pass in.

You specify a series of formatting substrings in `templateP`. Each substring has the form:

```
^<valueType><formatModifier>
```

Each substring has three components:

- The `^` character begins a substring.
- The `<valueType>` component is a single-digit value that specifies the value type.
- The `<formatModifier>` component is a single-letter value that specifies how you want that value formatted.

The following is an example of a template specification with three substrings:

```
^0z ^2l ^4r
```

Table 57.1 shows the values you can specify for the `<valueType>` component. Note that the formatted result depends on the `<modifier>` value.

Table 57.1 Template value types for the `DateTemplateToAscii` function

Value	Value type	Formatted examples
0	Day number	1, 01, 23, 31
1	Day name	Tue, Tuesday
2	Month name	May, Aug, August
3	Month number	4, 04, 11
4	Year number	97, 1997

Table 57.2 shows the values you can specify for the <modifier> component of each template substring.

Table 57.2 Template modifier types for the `DateTemplateToAscii` function

Modifier	Description
s	Formats the value in short form
r	Formats the value in regular form
l	Formats the value in long form
z	Adds a leading zero to the formatted numeric value

Finally, Table 57.3 shows examples of each value type formatted with each modifier type.

Table 57.3 Examples of formatted values

Value type	Raw value	`s` (Short format)	`r` (Regular format)	`l` (Long format)	`z` (Zero format)
0 (Day number)	2	2	2	2	02
1 (Day name)	2	T	Tue	Tuesday	n/a

Table 57.3 Examples of formatted values *(continued)*

Value type	Raw value	`s` (Short format)	`r` (Regular format)	`l` (Long format)	`z` (Zero format)
2 (Month name)	`11`	N	Nov	November	n/a
3 (Month number)	`11`	11	11	11	11
4 (Year number)	`2000`	00	2000	2000	n/a

For example, calling `DateTemplateToAscii` as follows:

```
DateTemplateToAscii("^0z ^2l ^4r", 2, 7,
  2000, myStr, 20)
```

Produces the following formatted string:

```
07 February 2000
```

Compatibility Implemented only if 3.5 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `DateGlueTemplateToAscii`. For more information, see Chapter 80, "PalmOSGlue Library."

See Also DateToAscii(), DateToDOWDMFormat()

DateToAscii Function

Purpose Convert the passed date to a string using the format specified by the `dateFormat` parameter.

Declared In `DateTime.h`

Prototype

```
void DateToAscii (UInt8 months, UInt8 days,
   UInt16 years, DateFormatType dateFormat,
   Char *pString)
```

Parameters

→ *months*
The month number, which must be a value between `1` and `12`.

→ *days*
The day number, which must be a value between `1` and `31`.

→ *years*
The four-digit year number. For example, `1995`.

→ *dateFormat*
Any `DateFormatType` format.

← *pString*
A pointer to string that is updated with the result.

This string must be of length `dateStringLength` for short formats or `longDateStrLength` for medium or long formats. Note that these lengths do include the terminating null byte. For more information about required string lengths, see Time Manager Constants.

Returns Returns nothing. The string reference by `pString` is updated with the formatted string.

Comments If you are using a debug ROM, the string buffer is filled with either `dateStringLength` or `longStrLength` debugging bytes, depending on the value of the `dateFormat` parameter.

It is important to allocate enough space for your string buffer. Finding buffer overflow errors can be difficult when using a debug ROM. One common situation is when you pass a buffer that is too small from a form, for an element such as a label or title. Then, the buffer overflow can clobber objects that follow the form in memory. When a form element's location information is corrupted, it disappears from the display.

Note that you can use the `DateTemplateToAscii()` function instead of this function if the 3.5 feature set is present. You can call the `DateTemplateToAscii()` function with a NULL string buffer to predetermine the required size for your buffer.

See Also TimeToAscii(), DateToDOWDMFormat(), DateTemplateToAscii()

DateToDays Function

Purpose Convert the `DateType` structure to the number of days elapsed from January 1, 1904.

Declared In `DateTime.h`

Prototype

```
UInt32 DateToDays (DateType date)
```

Parameters

→ *`date`*
A `DateType` structure.

Returns Returns the number of days elapsed from January 1, 1904 to the specified `date`.

See Also `DateDaysToDate()`

DateToDOWDMFormat Function

Purpose Convert a date to a formatted string using the format specified by the `dateFormat` parameter. The resultant string includes the name of the day of the week.

Declared In `DateTime.h`

Prototype

```
void DateToDOWDMFormat (UInt8 months, UInt8 days,
    UInt16 years, DateFormatType dateFormat,
    Char *pString)
```

Parameters

→ *`months`*
The month number, which must be a value between `1` and `12`.

→ *`days`*
The day number, which must be a value between `1` and `31`.

→ *`years`*
The four-digit year number. For example, `1995`.

→ *dateFormat*
Any DateFormatType format.

← *pString*
A pointer to a string that is updated with the result. The string must be of length dowDateStringLength for short formats or dowLongDateStrLength for medium or long date formats. See Time Manager Constants for string buffer lengths.

Returns Returns nothing. The string referenced by pString is updated with the formatted string.

Comments The values of some of the Time Manager Constants that specify the required string buffer lengths do change from time to time. You should always use the constants or verify the required lengths by checking the datetime.h file.

It is important to allocate enough space for your string buffer. Finding buffer overflow errors can be difficult when using a debug ROM. One common situation is when you pass a buffer that is too small from a form, for an element such as a label or title. Then, the buffer overflow can clobber objects that follow the form in memory. When a form element's location information is corrupted, it disappears from the display.

Note that you can use the DateTemplateToAscii() function instead of this function if the 3.5 feature set is present. You can call the DateTemplateToAscii() function with a NULL string buffer to predetermine the required size for your buffer.

Compatibility On Palm OS 3.1 Japanese ROMs, this function contains a bug that prevented it from properly displaying 4-byte long day names. To prevent this bug from affecting your application, use DateGlueToDOWDMFormat in the PalmOSGlue library instead of calling this function directly. For more information, see Chapter 80, "PalmOSGlue Library."

See Also DateToAscii(), DateTemplateToAscii()

DayOfMonth Function

Purpose Return a value that represents the day of a month on which the specified date occurs. The value represents a quantity such as "First

Monday" or "Third Friday" as is used for repeating appointments in the Datebook.

Declared In `DateTime.h`

Prototype

```
Int16 DayOfMonth (Int16 month, Int16 day,
    Int16 year)
```

Parameters

→ *month*
The month number, which must be a value between `1` and `12`.

→ *day*
The day number, which must be a value between `1` and `31`.

→ *year*
The four-digit year number. For example, `1995`.

Returns Returns a value that represents day of the month. This value is one of the `DayOfMonthType` values.

Comments The returns value can be used to specify on which day of the month an appointment repeats.

DayOfWeek Function

Purpose Return the day of the week value for a specified date.

Declared In `DateTime.h`

Prototype

```
Int16 DayOfWeek (Int16 month, Int16 day,
    Int16 year)
```

Parameters

→ *month*
The month number, which must be a value between `1` and `12`.

→ *day*
The day number, which must be a value between `1` and `31`.

→ *year*
The four-digit year number. For example, `1995`.

Returns Returns one of the following values for the day of the week of the specified date, as shown in the following table:

Day name	Returned day value
Sunday	0
Monday	1
Tuesday	2
Wednesday	3
Thursday	4
Friday	5
Saturday	6

DaysInMonth Function

Purpose Return the number of days in the month.

Declared In `DateTime.h`

Prototype

```
Int16 DaysInMonth (Int16 month, Int16 year)
```

Parameters

→ *month*
The month number, which must be a value between 1 and 12.

→ *year*
The four-digit year number. For example, 1995.

Returns Returns the number of days in the month for the specified year.

TimAdjust Function

Purpose Return a new date, with the time adjusted by the specified number of seconds.

Declared In `DateTime.h`

Prototype

```
void TimAdjust (DateTimeType *dateTimeP,
    Int32 adjustment)
```

Parameters

↔ *dateTimeP*
A pointer to a `DateType` structure.

→ *adjustment*
The number of seconds by which to adjust the time.

Returns Returns nothing. The structure referenced by `dateTimeP` is modified to contain the updated date and time.

Comments This function advances the time by the specified number of seconds and takes care of any wraparound conditions.

See Also `DateAdjust()`

TimDateTimeToSeconds Function

Purpose Return the number of seconds elapsed from 12:00 A.M. on January 1, 1904 to the specified date and time.

Declared In `DateTime.h`

Prototype

```
UInt32 TimDateTimeToSeconds
    (const DateTimeType *dateTimeP)
```

Parameters → *dateTimeP*
A pointer to a `DateTimeType` structure.

Returns The number of seconds elapsed from 12:00 A.M. on January 1, 1904 to the date referenced by `dateTimeP`.

See Also TimSecondsToDateTime()

TimGetSeconds Function

Purpose Return the current date and time of the device in seconds since 12:00 A.M. on January 1, 1904.

Declared In `TimeMgr.h`

Prototype

```
UInt32 TimGetSeconds (void)
```

Parameters `None.`

Returns The number of seconds elapsed from 12:00 A.M. on January 1, 1904 to the current date and time on the device.

See Also `TimSetSeconds()`

TimGetTicks Function

Purpose Return the tick count since the last reset. The tick count does not advance while the device is in sleep mode.

Declared In `TimeMgr.h`

Prototype

```
UInt32 TimGetTicks (void)
```

Parameters `None.`

Returns Returns the tick count.

Comments You can call the `SysTicksPerSecond()` routine to determine the number of ticks per second.

See Also `SysTicksPerSecond()`

TimSecondsToDateTime Function

Purpose Converts a date specified as the number of seconds since January 1, 1904 to a `DateTimeType` structure.

Declared In `DateTime.h`

Prototype

```
void TimSecondsToDateTime (UInt32 seconds,
    DateTimeType *dateTimeP)
```

Parameters

→ *seconds*
: A date specified as the number of seconds elapsed from 12:00 A.M. on January 1, 1904 to the date

← *dateTimeP*
: A pointer to a `DateTimeType` structure that is updated with the date and time values.

Returns Returns nothing. The structure referenced by `dateTimeP` is updated with the date and time computed for the number of seconds since 12:00 A.M. on January 1, 1904.

See Also `TimDateTimeToSeconds()`

TimSetSeconds Function

Purpose Set the clock of the device to the date and time passed as the number of seconds since 12:00 A.M. on January 1, 1904.

Declared In `TimeMgr.h`

Prototype

```
void TimSetSeconds (UInt32 seconds)
```

Parameters

→ *seconds*
The number of seconds since 12:00 A.M. on January 1, 1904.

Returns Returns nothing.

Comments If the Notification Feature Set is present, this function broadcasts the `sysNotifyTimeChangeEvent` to all interested parties. See Chapter 43, "Notification Manager," for more information.

See Also `TimGetSeconds()`

TimeToAscii Function

Purpose Convert the time to a string that is formatted according to the specified `timeFormat`.

Declared In `DateTime.h`

Prototype

```
void TimeToAscii (UInt8 hours, UInt8 minutes,
    TimeFormatType timeFormat, Char *pString)
```

Parameters

→ *hours*
The number of hours. This must be a value between `0` and `23`.

→ *minutes*
The number of minutes. This must be a value between `0` and `59`.

→ *timeFormat*
The time format for the resultant string. This must be one of the `TimeFormatType` values.

← *pString*
A pointer to a string that is updated with the resultant string. This string must be of length `timeStringLength`.

See Time Manager Constants for information on string buffer lengths.

Returns Returns nothing. The string referenced by `pString` is updated with the formatted string.

Comments If you are using a debug ROM in Palm OS 3.5, the string buffer is filled with `timeStringLength` debugging bytes.

It is important to allocate enough space for your string buffer. Finding buffer overflow errors can be difficult when using a debug ROM. One common situation is when you pass a buffer that is too small from a form, for an element such as a label or title. Then, the buffer overflow can clobber objects that follow the form in memory. When a form element's location information is corrupted, it disappears from the display.

See Also `DateToAscii()`

TimeZoneToAscii Function

Purpose Convert a time zone to a string.

Declared In `DateTime.h`

Prototype

```
void TimeZoneToAscii (Int16 timeZone,
   const LmLocaleType *localeP, Char *string)
```

Parameters

→ *timeZone*
: A pointer to the time zone, given as minutes east of Greenwich Mean Time (GMT).

→ *localeP*
: A pointer to a locale (see `LmLocaleType`) that identifies the time zone country. You can use the constant `lmAnyLanguage` as the value for the language field of the structure pointed to by this parameter.

← *string*
: A pointer to a string in which to return the result. This string must be of length `timeZoneStringLength`.

Returns Returns nothing.

Comments This function returns a descriptive string for the specified time zone. This string identifies the time zone first by its country, such as "USA (Mountain)" or "Canada (Eastern)." If the function cannot find a time zone that matches the specified GMT offset and country, it

returns a string containing the time zone as a numeric offset from the GMT (for example, "GMT+9:00").

Compatibility Implemented only if 4.0 New Feature Set is present.

TimTimeZoneToUTC Function

Purpose Converts a date and time from a given time zone to Universal Coordinated Time (UTC). UTC is also known as Greenwich Mean Time (GMT).

Declared In `DateTime.h`

Prototype

```
UInt32 TimTimeZoneToUTC (UInt32 seconds,
    Int16 timeZone,
    Int16 daylightSavingAdjustment)
```

Parameters

→ *seconds*
The number of seconds since 12:00 A.M. on January 1, 1904.

→ *timeZone*
The time zone, given as the number of minutes east of UTC. For time zones west of UTC but before the international dateline, this is a negative number.

→ *daylightSavingAdjustment*
The number of minutes to add to the current time for daylight savings time in this time zone.

Returns Returns the same time as `seconds` but in the Universal Coordinated Time. The value is still given as the number of seconds since 12:00 A.M. on January 1, 1904.

Comments The returned value is not necessarily the time in Greenwich because Greenwich may be observing daylight saving time.

You can use this function to convert the local time to UTC. The time zone and the daylight savings adjustment are system preferences that can be retrieved using `PrefGetPreference()`. For example, the following code converts the current local time to UTC:

```
Int16 timeZone =
  PrefGetPreference(prefTimeZone);
Int16 daylightSavingAdjustment =
  PrefGetPreference(
  prefDaylightSavingAdjustment);
```

```
UInt32 utcTime =
  TimTimeZoneToUTC(TimGetSeconds(), timeZone,
  daylightSavingAdjustment);
```

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TimUTCToTimeZone()`

TimUTCToTimeZone Function

Purpose Converts a date and time from Universal Coordinated Time (UTC) to the specified time zone. UTC is also known as Greenwich Mean Time (GMT).

Declared In `DateTime.h`

Prototype

```
UInt32 TimUTCToTimeZone (UInt32 seconds,
    Int16 timeZone,
    Int16 daylightSavingAdjustment)
```

Parameters

→ *seconds*
The number of seconds since 12:00 A.M. on January 1, 1904 in UTC.

→ *timeZone*
The time zone, given as the number of minutes east of UTC. For time zones west of UTC before the international dateline, this is a negative number.

→ *daylightSavingAdjustment*
The number of minutes to add to the current time for daylight savings time in this time zone.

Returns Returns the same time as `seconds` but in the specified time zone. The value is still given as the number of seconds since 12:00 A.M. on January 1, 1904.

Comments The `seconds` value is not necessarily the time in Greenwich because Greenwich may be observing daylight saving time.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TimTimeZoneToUTC()`

58

Virtual File System Manager

The Virtual File System (VFS) Manager is a layer of software that manages all installed file system libraries. It provides a unified API to application developers while allowing them to seamlessly access many different types of file systems —such as VFAT, HFS, and NFS—on many different types of media, including Compact Flash, Memory Stick, and SmartMedia.This chapter provides reference material for the VFS Manager API as follows:

- VFS Manager Data Structures
- VFS Manager Constants
- VFS Manager Functions
- Application-Defined Functions

The header file `VFSMgr.h` declares the VFS Manager API. For more information on the VFS Manager, see Chapter 7, "Expansion," in *Palm OS Programmer's Companion*, vol. I.

Note that the VFS Manager is an optional system extension; the functions described in this chapter are implemented only if the VFS Manager Feature Set is present.

VFS Manager Data Structures

FileInfoType Struct

Purpose The `FileInfoType` structure contains information about a specified file or directory. This information is returned as a parameter to `VFSDirEntryEnumerate()`. The structure is defined as follows:

Prototype

```
typedef struct FileInfoTag {
  UInt32 attributes;
  Char *nameP;
  UInt16 nameBufLen;
} FileInfoType, *FileInfoPtr;
```

Fields

`attributes`
: Characteristics of the file or directory. See File and Directory Attributes for the bits that make up this field.

`nameP`
: Pointer to the buffer that receives the full name of the file or directory. Initialize this parameter to `NULL` if you don't want to receive the name.

`nameBufLen`
: Size of the `nameP` buffer, in bytes.

FileRef Typedef

Purpose

The `FileRef` type is used to encode references to files and directories.

Prototype

```
typedef UInt32 FileRef;
```

VFSAnyMountParamType Struct

Purpose

The `VFSAnyMountParamType` structure is a base structure for `VFSSlotMountParamType`, `VFSPOSEMountParamType`, and other similar structures that may be defined in the future. Use one or the other according to how you set the `mountClass` parameter.

Prototype

```
typedef struct VFSAnyMountParamTag {
  UInt16 volRefNum;
  UInt16 reserved;
  UInt32 mountClass;
} VFSAnyMountParamType;
typedef VFSAnyMountParamType
*VFSAnyMountParamPtr;
```

Fields

`volRefNum`
: The volume reference number. This is initially obtained when you successfully mount a volume. It can then be used to

format a volume with `VFSVolumeFormat()` or unmount a volume with `VFSVolumeUnmount`.

`reserved`
Reserved for future use.

`mountClass`
Defines the type of mount to use with the specified volume. See Volume Mount Classes for a list of mount types.

VFSSlotMountParamType Struct

Purpose The `VFSSlotMountParamType` structure is used when you are mounting a card located in an Expansion Manager slot. The `vfsMountParam->mountClass` field must be set to `VFSMountClass_SlotDriver`.

Prototype

```
typedef struct VFSSlotMountParamTag {
  VFSAnyMountParamType vfsMountParam;
  UInt16 slotLibRefNum;
  UInt16 slotRefNum;
} VFSSlotMountParamType;
```

Fields

`vfsMountParam`
See the description of `VFSAnyMountParamType` for an explanation of the fields in this structure. Set `vfsMountParam->mountClass` to `VFSMountClass_SlotDriver` to mount an Expansion Manager slot.

`slotLibRefNum`
Reference number for the slot driver library allocated to the given slot number. Obtain this field by calling `ExpSlotLibFind()`.

`slotRefNum`
Number of the slot to be mounted.

VFSPOSEMountParamType Struct

Purpose The `VFSPOSEMountParamType` structure is used when you are mounting a volume through the Palm OS® Emulator. The `vfsMountParam->mountClass` must be set to

VFSMountClass_POSE. Note that ordinary applications and file systems shouldn't use VFSPOSEMountParamType.

Prototype

```
typedef struct VFSPOSEMountParamTag {
  VFSAnyMountParamType vfsMountParam;
  UInt8 poseSlotNum;
} VFSPOSEMountParamType;
```

Fields

vfsMountParam
: See the description of VFSAnyMountParamType for an explanation of the fields in this structure. Set vfsMountParam->mountClass to VFSMountClass_POSE to mount a virtual slot.

poseSlotNum
: Number of the virtual slot number to be mounted by the Palm OS Emulator.

VolumeInfoType Struct

Purpose

The VolumeInfoType structure defines information that is returned to VFSVolumeInfo() and used throughout the VFS functions.

Prototype

```
typedef struct VolumeInfoTag {
  UInt32 attributes;
  UInt32 fsType;
  UInt32 fsCreator;
  UInt32 mountClass;
  UInt16 slotLibRefNum;
  UInt16 slotRefNum;
  UInt32 mediaType;
  UInt32 reserved;
} VolumeInfoType, *VolumeInfoPtr;
```

Fields

attributes
: Characteristics of the volume. See Volume Attributes for the bits that make up this field.

fsType
: File system type for this volume. See Defined File Systems for a list of the supported file systems.

`fsCreator`
: Creator ID of this volume's file system driver. This information is used with `VFSCustomControl()`.

`mountClass`
: Mount class that mounted this volume. The supported mount classes are listed under Volume Mount Classes.

`slotLibRefNum`
: Reference to the slot driver library with which the volume is mounted. This field is only valid when the mount class is `vfsMountClass_SlotDriver`.

`slotRefNum`
: Slot number where the card containing the volume is loaded. This field is only valid when the mount class is `vfsMountClass_SlotDriver`.

`mediaType`
: Type of card media. See Defined Media Types in the Expansion Manager chapter for the list of values. This field is only valid when the mount class is `vfsMountClass_SlotDriver`.

`reserved`
: Reserved for future use.

VFS Manager Constants

Defined File Systems

The following file systems are currently defined by the VFS Manager. These values are used with `VFSVolumeInfo()` in the `VolumeInfoType.fsType` parameter.

Constant	Value	Description
`vfsFilesystemType_AFS`	`'afsu'`	Unix Andrew file system
`vfsFilesystemType_EXT2`	`'ext2'`	Linux file system

Constant	Value	Description
`vfsFilesystemType_FAT`	`'fats'`	FAT12 and FAT16, which only handles 8.3 filenames
`vfsFilesystemType_FFS`	`'ffsb'`	Unix Berkeley block based file system
`vfsFilesystemType_HFS`	`'hfss'`	Macintosh standard hierarchical file system
`vfsFilesystemType_HFSPlus`	`'hfse'`	Macintosh extended hierarchical file system
`vfsFilesystemType_HPFS`	`'hpfs'`	OS2 High Performance file system
`vfsFilesystemType_MFS`	`'mfso'`	Macintosh original file system
`vfsFilesystemType_NFS`	`'nfsu'`	Unix Networked file system
`vfsFilesystemType_Novell`	`'novl'`	Novell file system
`vfsFilesystemType_NTFS`	`'ntfs'`	Windows NT file system
`vfsFilesystemType_VFAT`	`'vfat'`	FAT12 and FAT16 extended to handle long filenames

Open Mode Constants

This section describes constants that are used for the `openMode` parameter to the `VFSFileOpen()` function. These constants specify the mode in which a file or directory is opened.

Constant	Value	Description
`vfsModeExclusive`	`0x0001U`	Open and lock the file or directory. This mode excludes anyone else from using the file or directory until it is closed.
`vfsModeRead`	`0x0002U`	Open for read access.
`vfsModeWrite`	`0x0004U | vfsModeExclusive`	Open for write access.
`vfsModeReadWrite`	`vfsModeWrite | vfsModeRead`	Open for read/write access.
`vfsModeCreate`	`0x0008U`	Create the file if it doesn't already exist. This open mode is implemented in the VFS layer, rather than in the file system library.
`vfsModeTruncate`	`0x0010U`	Truncate the file to zero (0) bytes after opening, removing all existing data. This open mode is implemented in the VFS layer, rather than in the file system library.
`vfsModeVFSLayerOnly`	`vfsModeCreate | vfsModeTruncate | vfsModeLeaveOpen`	Mask used to isolate those flags that are only used by the VFS layer. These flags are not passed to the file system layer.
`vfsModeLeaveOpen`	`0x0020U`	Leave the file open even after the application exits.

File and Directory Attributes

The constants in the following table define bits that can be used individually or in combination when setting or interpreting the file attributes for a given file or directory. See

VFSFileGetAttributes(), VFSFileSetAttributes(), and the FileInfoType data structure for specific use.

Constant	Value	Description
vfsFileAttrReadOnly	0x00000001UL	Read-only file or directory
vfsFileAttrHidden	0x00000002UL	Hidden file or directory
vfsFileAttrSystem	0x00000004UL	System file or directory
vfsFileAttrVolumeLabel	0x00000008UL	Volume label
vfsFileAttrDirectory	0x00000010UL	Directory
vfsFileAttrArchive	0x00000020UL	Archived file or directory
vfsFileAttrLink	0x00000040UL	Link to another file or directory

Volume Attributes

The constants in the following table define bits that can be used individually or in combination to make up the attributes field in the VolumeInfoType structure.

Constant	Value	Description
vfsVolumeAttrHidden	0x00000004UL	The volume should not be visible to the user.
vfsVolumeAttrNonRemovable	0x00000008UL	The media cannot be removed.
vfsVolumeAttrReadOnly	0x00000002UL	The volume is read only.
vfsVolumeAttrSlotBased	0x00000001UL	Reserved. Check the mount class to determine how a volume is mounted.

Volume Mount Classes

The following constants define how a given volume is mounted. The mountClass field in the VFSAnyMountParamType and VolumeInfoType structures takes on one of these values.

Constant	Value	Description
`vfsMountClass_POSE`	`'pose'`	Mount the volume through the Palm OS Emulator. This is used for testing.
`vfsMountClass_Simulator`	`sysFileTSimulator`	Mount the volume through the simulator. This is used for testing.
`vfsMountClass_SlotDriver`	`sysFileTSlotDriver`	Mount the volume with a slot driver shared library.

Error Codes

The VFS Manager defines the following error codes:

Constant	Description
`vfsErrBadData`	The operation could not be completed because of invalid data.
`vfsErrBadName`	Invalid filename, path, or volume label.
`vfsErrBufferOverflow`	The supplied buffer is too small.
`vfsErrDirectoryNotFound`	Returned when the path leading up to the file does not exist.
`vfsErrDirNotEmpty`	The directory is not empty and therefore cannot be deleted.
`vfsErrFileAlreadyExists`	A file with this name exists already in this location.

Constant	Description
`vfsErrFileBadRef`	The file reference is invalid: it has been closed or was not obtained from `VFSFileOpen()`.
`vfsErrFileEOF`	The file pointer is at the end of the file.
`vfsErrFileGeneric`	Generic file error.
`vfsErrFileNotFound`	The file was not found at the specified location.
`vfsErrFilePermissionDenied`	The requested permissions could not be granted.
`vfsErrFileStillOpen`	Returned from the underlying file system's delete function if the file is still open.
`vfsErrIsADirectory`	This operation can only be performed on a regular file, not a directory.
`vfsErrNameShortened`	A volume name or filename was automatically shortened to conform to the file system specification.
`vfsErrNoFileSystem`	None of the installed file systems support this operation.
`vfsErrNotADirectory`	This operation can only performed on a directory.
`vfsErrVolumeBadRef`	The volume reference number is invalid.

Constant	Description
`vfsErrVolumeFull`	There is insufficient space left on the volume.
`vfsErrVolumeStillMounted`	Returned from the underlying file system's format function if the volume is still mounted.

VFS Manager Functions

VFSCustomControl Function

Purpose Make a custom API call to a particular file system, given its creator ID. You can use `VFSVolumeInfo()` to determine the creator ID of the file system for a given volume.

Declared In `VfsMgr.h`

Prototype

```
Err VFSCustomControl(UInt32 fsCreator,
   UInt32 apiCreator, UInt16 apiSelector,
   void *valueP, UInt16 *valueLenP)
```

Parameters

→ *`fsCreator`*
Creator of the file system to call. A value of zero (0) tells the VFS Manager to check each registered file system, looking for one which supports the call.

→ *`apiCreator`*
Registered creator ID.

→ *`apiSelector`*
Custom operation to perform.

↔ *`valueP`*
A pointer to a buffer containing data specific to the operation. On exit, depending on the function of the particular custom call and on the value of `valueLenP`, the contents of this buffer may have been updated.

↔ *`valueLenP`*
On entry, points to the size of the `valueP` buffer. On exit, this value reflects the size of the data written to the `valueP`

buffer. If `valueLenP` is `NULL`, `valueP` is passed to the file system but is not updated on exit.

Returns Returns one of the following error codes:

`errNone`
No error.

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`expErrUnsupportedOperation`
The specified opcode and/or creator is unsupported or undefined.

`sysErrParamErr`
The `valueP` buffer is too small.

`vfsErrNoFileSystem`
VFS Manager cannot find an appropriate file system to handle the request.

Comments The driver identifies the call and its API by a registered creator ID and a selector. This allows file system developers to extend the API by defining selectors for their creator IDs. It also allows file system developers to support selectors (and custom calls) defined by other file system developers.

This function must return `expErrUnsupportedOperation` for all unsupported or undefined opcodes and/or creators.

Compatibility Implemented only if the VFS Manager Feature Set is present.

VFSDirCreate Function

Purpose Create a new directory.

Declared In `VfsMgr.h`

Prototype

```
Err VFSDirCreate(UInt16 volRefNum,
    const Char *dirNameP)
```

Parameters → *`volRefNum`*
Volume reference number returned from `VFSVolumeEnumerate()`.

→ *dirNameP*
Pointer to the full path of the directory to be created.

Returns Returns one of the following error codes:

`errNone`
No error

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`vfsErrBadName`
Some or all of the path, up to but not including the last component specified in the `dirNameP` parameter, does not exist.

`vfsErrFileAlreadyExists`
A file with this name already exists in this location.

`vfsErrNoFileSystem`
The VFS Manager cannot find an appropriate file system to handle the request.

`vfsErrVolumeBadRef`
The volume has not been mounted.

`vfsErrVolumeFull`
There is not enough space left on the volume.

Comments All parts of the path except the last component must already exist. The `vfsFileAttrDirectory` attribute is set with this function.

`VFSDirCreate()` does not open the directory. Any operations you want to perform on this directory require a reference, which is obtained through a call to `VFSFileOpen()`.

Compatibility Implemented only if the VFS Manager Feature Set is present.

VFSDirEntryEnumerate Function

Purpose Enumerate the entries in a given directory. Entries can include files, links, and other directories.

Declared In `VfsMgr.h`

Prototype

```
Err VFSDirEntryEnumerate (FileRef dirRef,
    UInt32 *dirEntryIteratorP, FileInfoType
    *infoP)
```

Parameters

→ *dirRef*
: Directory reference returned from `VFSFileOpen()`.

↔ *dirEntryIteratorP*
: Pointer to the index of the last entry enumerated. For the first iteration, initialize this parameter to the constant `vfsIteratorStart`. Upon return, this references the next entry in the directory. If `infoP` is the last entry, this parameter is set to `vfsIteratorStop`.

↔ *infoP*
: Pointer to the `FileInfoType` data structure that contains information about the given directory entry. The `nameP` and `nameBufLen` fields in this structure must be initialized prior to calling `VFSDirEntryEnumerate`.

Returns Returns one of the following error codes:

`errNone`
: No error.

`expErrEnumerationEmpty`
: There are no directory entries left to enumerate.

`expErrNotOpen`
: The file system library necessary for this call has not been installed or has not been opened.

`sysErrParamErr`
: The `dirEntryIteratorP` is not valid.

`vfsErrFileBadRef`
: The specified file reference is invalid.

`vfsErrIsNotADirectory`
: The specified file reference is valid, but does not point to a directory.

`vfsErrNoFileSystem`
: The VFS Manager cannot find an appropriate file system to handle the request.

Comments

The directory to be enumerated must first be opened with `VFSFileOpen()` in order to obtain a file reference. In order to obtain information on all entries in a directory you must make repeated calls to `VFSDirEntryEnumerate` inside a loop. Boundaries on the iteration are the defined constants `vfsIteratorStart` and `vfsIteratorStop`. Before the first call to `VFSDirEntryEnumerate`, `dirEntryIteratorP` should be initialized to `vfsIteratorStart`. Each iteration then changes the value pointed to by `dirEntryIteratorP`. When information on the last entry in the directory is returned, `dirEntryIteratorP` is set to `vfsIteratorStop`.

WARNING! Creating, renaming, or deleting any file or directory invalidates the enumeration. After any such operation, the enumeration will need to be restarted.

Example

The following code excerpt illustrates how to use `VFSDirEntryEnumerate`.

```
FileInfoType info;
FileRef dirRef;
UInt32 dirIterator;
Char *fileName = MemPtrNew(256);  // should check for err

// open the directory first, to get the directory reference
// volRefNum must have already been defined
err = VFSFileOpen(volRefNum, "/", vfsModeRead, &dirRef);
if(err == errNone) {

   info.nameP = fileName;    // point to local buffer
   info.nameBufLen = 256;
   dirIterator = vfsIteratorStart
   while (dirIterator != vfsIteratorStop) {
      // Get the next file
      err = VFSDirEntryEnumerate (dirRef, &dirIterator,
         &info);
      if (err == errNone) {
         // Do something with the directory entry information
         // Pull the attributes from info.attributes
         // The file name is in fileName
```

```
      } else {
         // handle error, possibly by breaking out of the
loop
      }
   } else {
      // handle directory open error here
   }
   MemPtrFree(fileName);
}
```

Compatibility Implemented only if the VFS Manager Feature Set is present.

VFSExportDatabaseToFile Function

Purpose Save the specified database to a `.pdb` or .`prc` file on an external storage card.

Declared In `VfsMgr.h`

Prototype

```
Err VFSExportDatabaseToFile (UInt16 volRefNum,
   const Char *pathNameP, UInt16 cardNo,
   LocalID dbID)
```

Parameters

→ *`volRefNum`*
Volume on which the destination file should be created.

→ *`pathNameP`*
Pointer to the complete path and name of the destination file to be created.

→ *`cardNo`*
Card number on which the `.pdb` or `.prc` being exported resides.

→ *`dbID`*
ID of the database being exported.

Returns Returns one of the following error codes:

`errNone`
No error

`expErrNotEnoughPower`
There is insufficient battery power to perform the database export operation.

`vfsErrBadName`
The path name specified in `pathNameP` is not valid.

Comments This utility function exports a database from main memory to a `.pdb` or `.prc` file on an external storage card. This function is the opposite of `VFSImportDatabaseFromFile()`. It first creates the file specified in the `pathNameP` parameter with `VFSFileCreate()`. After opening the file the Exchange Manager function `ExgDBWrite()` is called with an internal callback function for exporting the file from the Data Manager. The Exchange Manager makes repeated calls to this callback function, which receives the data back in blocks. Once all the data has been exported, VFS Manager closes the file.

This function is used, for example, to copy applications from main memory to a storage card.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also `VFSExportDatabaseToFileCustom()`, `VFSFileWrite()`, `VFSImportDatabaseFromFile()`

VFSExportDatabaseToFileCustom Function

Purpose Saves the specified database to a .pdb or .prc file on an external storage card. This function differs from `VFSExportDatabaseToFile()` in that it allows you to track the progress of the export operation.

Declared In `VfsMgr.h`

Prototype

```
Err VFSExportDatabaseToFileCustom
    (UInt16 volRefNum, const Char *pathNameP,
    UInt16 cardNo, LocalID dbID,
    VFSExportProcPtr exportProcP, void *userDataP)
```

Parameters

→ *volRefNum*
Volume on which the destination file should be created.

→ *pathNameP*
Pointer to the complete path and name of the destination file to be created.

→ *cardNo*
Card number on which the `.pdb` or `.prc` being exported resides.

→ *dbID*
ID of the database being exported.

→ *exportProcP*
User-defined callback function that tracks the progress of the export. This function should allow the user to cancel the export. Pass `NULL` if you don't have a progress callback function. See `VFSExportProcPtr()` for the requirements of this function.

→ *userDataP*
Pointer to any data you want to pass to the callback function specified in `exportProcP`. This information is not used internally by the VFS Manager. Pass `NULL` if you don't have a progress callback function or if that function doesn't need any such data.

Returns Returns one of the following error codes:

`errNone`
No error.

`expErrNotEnoughPower`
There is insufficient battery power to perform the database export operation.

`vfsErrBadName`
The path name specified in `pathNameP` is not valid.

This function can also return any error code other than `errNone` produced by your callback function.

Comments This function is similar to `VFSExportDatabaseToFile()` in that it exports a database from main memory to a `.pdb` or `.prc` file on an external storage card. It extends the functionality by allowing you to specify a callback function that tracks the progress of the export. It first creates the file specified in the `pathNameP` parameter with `VFSFileCreate()`. After opening the file, the Exchange Manager function `ExgDBWrite()` is called with an internal callback function for exporting the file from the Data Manager. Exchange Manager makes repeated calls to this function, which receives the data back in blocks. The progress tracker, if one has been specified, is also called every time a new chunk of data is

passed back. Once all the data has been exported, the VFS Manager closes the file.

This function is used, for example, to copy applications from main memory to a storage card.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also `VFSExportDatabaseToFile()`, `VFSFileWrite()`, `VFSImportDatabaseFromFileCustom()`

VFSFileClose Function

Purpose Closes a file or directory that has been opened with `VFSFileOpen()`.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileClose (FileRef fileRef)
```

Parameters

→ *fileRef*
File reference number returned from `VFSFileOpen()`.

Returns Returns one of the following error codes:

`errNone`
No error.

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`vfsErrFileBadRef`
The specified file reference is invalid.

Compatibility Implemented only if the VFS Manager Feature Set is present.

VFSFileCreate Function

Purpose Create a file. This function cannot be used to create a directory; use `VFSDirCreate()` instead.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileCreate(UInt16 volRefNum,
    const Char *pathNameP)
```

Parameters

→ *volRefNum*
: Reference number of the volume on which to create the file. This volume reference number is returned from `VFSVolumeEnumerate()`.

→ *pathNameP*
: Pointer to the full path of the file to be created. All parts of the path, excluding the filename, must already exist.

Returns Returns one of the following error codes:

`errNone`
: No error.

`expErrNotOpen`
: The file system library necessary for this call has not been installed or has not been opened.

`vfsErrBadName`
: The `pathNameP` is invalid.

`vfsErrFileAlreadyExists`
: A file with this name already exists in this location.

`vfsErrNoFileSystem`
: The VFS Manager cannot find an appropriate file system to handle the request.

`vfsErrVolumeBadRef`
: The volume has not been mounted.

`vfsErrVolumeFull`
: There is not enough space left on the volume.

Comments It is the responsibility of the file system library to ensure that all filenames are translated into a format that is compatible with the native format of the file system, such as the 8.3 convention for a FAT file system without long filename support. See Naming Files in the

Expansion chapter of the *Palm OS Programmer's Companion*, vol. I for a description of how to construct a valid path.

This function does not open the file. Use `VFSFileOpen()` to open the file.

This function should not be used to create a directory. To create a directory use `VFSDirCreate()`.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also `VFSFileDelete()`

VFSFileDBGetRecord Function

Purpose Load a record from an opened `.pdb` file on an external card into the storage heap.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileDBGetRecord (FileRef ref,
    UInt16 recIndex, MemHandle *recHP,
    UInt8 *recAttrP, UInt32 *uniqueIDP)
```

Parameters

→ *ref*
The file reference returned from `VFSFileOpen()`. Note that the open file must be a `.pdb` file.

→ *recIndex*
The index of the record to load.

← *recHP*
Pointer to the record data's handle in the storage heap. If `NULL` is returned in this parameter there is either no data in this field or an error occurred reading this data from the file. If the handle is not `NULL`, you must dispose of the allocated handle using `MemHandleFree()`.

← *recAttrP*
Pointer to the attributes of the record. The values returned are identical to the atttributes returned from `DmRecordInfo()`. See Record Attribute Constants in the Data and Resource Manager chapter for a description of each attribute. Pass `NULL` for this parameter if you do not want to retrieve this information.

← *uniqueIDP*
Pointer to the unique identifier for this record. Pass `NULL` for this parameter if you do not want to retrieve this information.

Returns Returns one of the following error codes:

`errNone`
No error.

`dmErrIndexOutOfRange`
The `recIndex` is out of range.

`dmErrNotRecordDB`
The file referenced by `ref` is not a record database.

`memErrNotEnoughSpace`
There is not enough space in memory for the requested record entry.

`sysErrParamErr`
A `NULL` value was passed in for the `recHP`, `recAttrP`, and `uniqueIDP` parameters.

`vfsErrBadData`
The local offsets (`localChunkID`) from the top of the `.pdb` to the start of the raw record data for this entry are out of order.

Comments This function is analogous to `DmGetRecord()` but works with files on an external card rather than databases in main memory. This function allocates a handle of the appropriate size from the storage heap and returns it in the `recHP` parameter. The caller is responsible for freeing this memory, using `MemHandleFree()`, when it is no longer needed.

NOTE: This function is not efficient for multiple accesses and should be used sparingly.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also `VFSFileReadData()`

VFSFileDBGetResource Function

Purpose Load a resource into the storage heap from an opened `.prc` file.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileDBGetResource (FileRef ref,
    DmResType type, DmResID resID, MemHandle
    *resHP)
```

Parameters

→ *ref*
The file reference returned from `VFSFileOpen()`. Note that the open file must be a `.prc` file.

→ *type*
The type of resource to load. See the Data and Resource Manager chapter for more information on resources.

→ *resID*
The ID of resource to load.

← *resHP*
Pointer to the resource data handle that was loaded into memory.

Returns Returns one of the following error codes:

`errNone`
No error.

`dmErrNotResourceDB`
The file referenced by `ref` is not a resource database.

`dmErrResourceNotFound`
The requested resource was not found.

`memErrNotEnoughSpace`
There is not enough space in memory for the requested resource entries.

`sysErrParamErr`
`resHP` is `NULL`.

Comments This function locates the specified resource in the open `.prc` file. See the *Palm OS File Format Specification* for more information on the layout of `.prc` files.

Once the resource is found, `VFSFileDBGetResource` allocates a handle of the appropriate size in the storage heap and reads it into memory. The handle to this memory location is returned through

the `resHP` parameter. The caller is responsible for freeing this memory, using `MemHandleFree()`, when it is no longer needed.

NOTE: This function is not efficient for multiple accesses and should be used sparingly.

Compatibility Implemented only if the VFS Manager Feature Set is present.

VFSFileDBInfo Function

Purpose Get information about a database represented by an open `.prc` or `.pdb` file.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileDBInfo (FileRef ref, Char *nameP,
    UInt16 *attributesP, UInt16 *versionP,
    UInt32 *crDateP, UInt32 *modDateP,
    UInt32 *bckUpDateP, UInt32 *modNumP,
    MemHandle *appInfoHP, MemHandle *sortInfoHP,
    UInt32 *typeP, UInt32 *creatorP,
    UInt16 *numRecordsP)
```

Parameters

→ *ref*
The file reference returned from `VFSFileOpen()`. Note that the open file must be a `.prc` or `.pdb` file.

← *nameP*
Pointer to a 32-byte character array in which the database name is returned. Pass `NULL` for this parameter if you do not want to retrieve the database name.

← *attributesP*
Pointer to the database attributes stored in the file. The values returned are identical to the atttributes returned from `DmDatabaseInfo()`. See the Database Attribute Constants section for a description of each attribute. Pass `NULL` for this parameter if you do not want to retrieve the database's attributes.

← *versionP*
Pointer to the application-specific version number of the database. The default version number is zero (0). Pass `NULL`

for this parameter if you do not want to retrieve the version number.

← *crDateP*
Pointer to the date the database was created, expressed in seconds since midnight (00:00:00) January 1, 1904. Pass `NULL` for this parameter if you do not want to retrieve the creation date.

← *modDateP*
Pointer to the date the database was last modified, expressed in seconds since midnight (00:00:00) January 1, 1904. A database's modification date is updated only if a change has been made to the database when it is opened with write access. Pass `NULL` for this parameter if you do not want to retrieve the database's modification date.

← *bckUpDateP*
Pointer to the date the database was last backed up, expressed in seconds since midnight (00:00:00) January 1, 1904. Pass `NULL` for this parameter if you do not want to retrieve the database's backup date.

← *modNumP*
Pointer to the number of times the database was modified. This number is updated every time a record is added, modified, or deleted. Pass `NULL` for this parameter if you do not want to retrieve the modification count.

← *appInfoHP*
Pointer to the application info block handle. If `NULL` is returned in this parameter, either there is no data in this field or an error occurred reading this data from the file. If a value other than `NULL` is returned, you must dispose of the allocated handle using `MemHandleFree()`. If you do not want to retrieve the application info block, pass `NULL` for this parameter

← *sortInfoHP*
Pointer to the sort info block handle. If `NULL` is returned in this parameter, either there is no data in this field or an error occurred reading this data from the file. If a value other than `NULL` is returned, you must dispose of the allocated handle using `MemHandleFree()`. Pass `NULL` for this parameter if you do not want to retrieve the sort info block handle.

← *typeP*
Pointer to the type of database as it was created. This may be a user-defined database type or a database type defined by the Palm OS. Some of the more common database types returned here are:

Type	Description
`'appl'`	Standard Palm™ application (resource database)
`'libr'`	Standard shared library
`'libf'`	File system shared library
`'libs'`	Slot driver shared library
`'data'`	Standard Palm data file (record database)

Pass `NULL` for this parameter if you do not want to retrieve the database's type.

← *creatorP*
Pointer to the database's creator. Pass NULL for this parameter if you do not want to retrieve this information.

← *numRecordsP*
Pointer to the number of records in the database. Pass `NULL` for this parameter if you do not want to retrieve this information.

Returns Returns one of the following error codes:

`errNone`
No error

`memErrNotEnoughSpace`
There is not enough space in memory for the database header.

`vfsErrBadData`
The file referenced by the `ref` parameter is too small to contain a database header, or the database header is corrupted.

Comments This function is analogous to `DmDatabaseInfo()`, but it works with files on an external card rather than with databases in main memory. See the *Palm OS File Format Specification* for a description of the header block in `.prc` and `.pdb` files.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also `VFSFileGetAttributes()`, `VFSFileGetDate()`

VFSFileDelete Function

Purpose Deletes a closed file or directory.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileDelete(UInt16 volRefNum,
    const Char *pathNameP)
```

Parameters

→ *volRefNum*
Volume reference number returned from `VFSVolumeEnumerate()`.

→ *pathNameP*
Pointer to the full path of the file or directory to be deleted.

Returns Returns one of the following error codes:

`errNone`
No error.

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`vfsErrBadName`
The path name specified in `pathNameP` is not valid.

`vfsErrDirNotEmpty`
The directory being deleted is not empty.

`vfsErrFileStillOpen`
The file is still open.

`vfsErrFileNotFound`
The file could not be found.

`vfsErrFilePermissionDenied`
The requested permissions could not be granted.

`vfsErrNoFileSystem`
The VFS Manager cannot find an appropriate file system to handle the request.

`vfsErrVolumeBadRef`
The volume has not been mounted.

Compatibility Implemented only if the VFS Manager Feature Set is present.

VFSFileEOF Function

Purpose Get end-of-file status for an open file. This function only operates on files and cannot be used with directories.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileEOF (FileRef fileRef)
```

Parameters → *`fileRef`*
File reference returned from `VFSFileOpen()`.

Returns Returns one of the following error codes:

`errNone`
No error. File pointer is not at end of the file.

`vfsErrFileEOF`
The file pointer is at the end of file.

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`vfsErrFileBadRef`
The specified file reference is invalid.

`vfsErrIsADirectory`
The specified file reference points to a directory instead of a file. This is an invalid operation on a directory.

`vfsErrNoFileSystem`
The VFS Manager cannot find an appropriate file system to handle the request.

Compatibility Implemented only if the VFS Manager Feature Set is present.

VFSFileGetAttributes Function

Purpose Obtain the attributes of an open file or directory.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileGetAttributes (FileRef fileRef,
   UInt32 *attributesP)
```

Parameters

→ *fileRef*
File reference returned from `VFSFileOpen()`.

← *attributesP*
Pointer to the attributes associated with the file or directory. See File and Directory Attributes for a list of values that can be returned through this parameter.

Returns Returns one of the following error codes:

`errNone`
No error

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`vfsErrFileBadRef`
The specified file reference is invalid.

`vfsErrNoFileSystem`
The VFS Manager cannot find an appropriate file system to handle the request.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSFileDBInfo(), VFSFileGetDate(), VFSFileSetAttributes()

VFSFileGetDate Function

Purpose Obtain the dates on an open file or directory.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileGetDate (FileRef fileRef,
   UInt16 whichDate, UInt32 *dateP)
```

Parameters

→ *fileRef*
File reference returned from `VFSFileOpen()`.

→ *whichDate*
Specifies which date—creation, modification, or last access—you want. Supply one of the following values:

`vfsFileDateCreated`

`vfsFileDateModified`

`vfsFileDateAccessed`
Note that not all file systems are required to support the above dates. If the supplied date type is not supported by the file system, `VFSFileGetDate` returns `expErrUnsupportedOperation`.

← *dateP*
Pointer to the requested date. This field is expressed in the standard Palm OS date format — the number of seconds since midnight (00:00:00) January 1, 1904.

Returns Returns one of the following error codes:

`errNone`
No error.

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`expErrUnsupportedOperation`
The specified date type is not supported by the underlying file system.

`vfsErrFileBadRef`
The specified file reference is invalid.

`sysErrParamErr`
The `whichDate` parameter is not one of the defined constants.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSFileDBInfo(), VFSFileGetAttributes(), VFSFileSetDate()

VFSFileOpen Function

Purpose Opens a file or directory and returns a reference for it.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileOpen (UInt16 volRefNum,
    const Char *pathNameP, UInt16 openMode,
    FileRef *fileRefP)
```

Parameters

→ *volRefNum*
: The volume reference number returned from `VFSVolumeEnumerate()`.

→ *pathNameP*
: Pointer to the full path of the file or directory to be opened. This must be a valid path. It cannot be empty and can not contain null characters. The format of the pathname should match what the underlying file system supports. See "Naming Files" in Chapter 7, "Expansion," of the *Palm OS Programmer's Companion*, vol. I for a description of how to construct a valid path.

→ *openMode*
: Mode to use when opening the file. See the Open Mode Constants section for a list of accepted modes.

← *fileRefP*
: Pointer to the opened file or directory reference which is supplied to various other `VFSFile...` operations. This value is filled in on return.

Returns Returns one of the following error codes:

`errNone`
: No error.

`expErrCardReadOnly`
: The open mode requested includes write access but the file is read-only.

`expErrNotOpen`
: The file system library necessary for this call has not been installed or has not been opened.

`vfsErrBadName`
: The `pathNameP` parameter is invalid.

`vfsErrFileNotFound`
The specified file or directory could not be found.

`vfsErrFilePermissionDenied`
The file cannot be opened in the requested open mode, or it has already been opened with `vfsModeExclusive`.

`vfsErrVolumeBadRef`
The specified volume has not been mounted.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also `VFSFileClose()`, `VFSDirEntryEnumerate()`

VFSFileRead Function

Purpose Read data from a file into the dynamic heap. This function only operates on files and cannot be used with directories; use `VFSDirEntryEnumerate()` to explore the contents of a directory.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileRead (FileRef fileRef, UInt32 numBytes,
    void *bufP, UInt32 *numBytesReadP)
```

Parameters

→ *fileRef*
File reference returned from `VFSFileOpen()`.

→ *numBytes*
Number of bytes to read.

← *bufP*
Pointer to the destination chunk where the data is to be stored. This can be a pointer to any writable memory.

← *numBytesReadP*
Pointer to an unsigned integer that reflects the number of bytes actually read. This value is set on return and does not need to be initialized. If no bytes are read the value is set to zero. Pass `NULL` for this parameter if you do not need to know how many bytes were read.

Returns Returns one of the following error codes:

`errNone`
No error.

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`vfsErrFileBadRef`
The specified file reference is invalid.

`vfsErrFileEOF`
The end of the file has been reached.

`vfsErrFilePermissionDenied`
Read permission is not enabled for this file.

`vfsErrIsADirectory`
The specified file reference is for a directory instead of a file. This is an invalid operation on a directory.

`vfsErrNoFileSystem`
The VFS Manager cannot find an appropriate file system to handle the request.

Comments The file system does not use `DmWrite()` and cannot be used to read data into the storage heap.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSFileReadData(), VFSFileWrite(), VFSImportDatabaseFromFile()

VFSFileReadData Function

Purpose Read data from a file into a chunk of memory in the storage heap. This function only operates on files and cannot be used with directories; use `VFSDirEntryEnumerate()` to explore the contents of a directory.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileReadData (FileRef fileRef,
   UInt32 numBytes, void *bufBaseP, UInt32
   offset, UInt32 *numBytesReadP)
```

Parameters

→ *fileRef*
File reference returned in `VFSFileOpen()`.

→ *numBytes*
Number of bytes to read.

← *bufBaseP*
: Pointer to the destination chunk in the storage heap where the data is to be stored. This pointer must be obtained through the appropriate call to the Memory Manager API.

→ *offset*
: Offset, in bytes, within the `bufBaseP` chunk where the data is to be written.

← *numBytesReadP*
: Pointer to an unsigned integer that reflects the number of bytes actually read. This value is set on return and does not need to be initialized. If no bytes are read, the value is set to zero. Pass `NULL` for this parameter if you do not need to know how many bytes were read.

Returns Returns one of the following error codes:

`errNone`
: No error.

`expErrNotOpen`
: The file system library necessary for this call has not been installed or has not been opened.

`vfsErrFileBadRef`
: The specified file reference is invalid.

`vfsErrFileEOF`
: The end of the file has been reached.

`vfsErrFilePermissionDenied`
: Read permission is not enabled for this file.

`vfsErrIsADirectory`
: The specified file reference is for a directory instead of a file. This is an invalid operation on a directory.

`vfsErrNoFileSystem`
: The VFS Manager cannot find an appropriate file system to handle the request.

Comments When data is read from an external card with `VFSFileReadData`, it is copied into a chunk of memory in the storage heap. This chunk *must* be allocated by the application before the call to `VFSFileReadData()`. This function calls `DmWrite()` to put the data in the storage heap.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSFileRead(), VFSFileWrite()

VFSFileRename Function

Purpose Rename a closed file or directory. This function cannot be used to move a file to another directory within the file system.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileRename (UInt16 volRefNum,
    const Char *pathNameP, const Char *newNameP)
```

Parameters

→ *volRefNum*
: Volume reference number returned from `VFSVolumeEnumerate()`.

→ *pathNameP*
: Pointer to the full path of the file or directory to be renamed.

→ *newNameP*
: Pointer to the new filename. Note that this is the name of the file only and does not include the path to the file.

Returns Returns one of the following error codes:

`errNone`
: No error.

`expErrNotOpen`
: The file system library necessary for this call has not been installed or has not been opened.

`vfsErrBadName`
: The name provided in either `pathNameP` or `newNameP` is invalid. This is also returned if the string pointed to by `newNameP` is a path, rather than a filename.

`vfsErrFileAlreadyExists`
: A file with the new name already exists in this location.

`vfsErrFileNotFound`
: The source file could not be found.

`vfsErrFilePermissionDenied`
: Write permission is not enabled for this file.

`vfsErrNoFileSystem`
: The VFS Manager cannot find an appropriate file system to handle the request.

`vfsErrVolumeBadRef`
: The volume has not been mounted.

`vfsErrVolumeFull`
: There is not enough space left on the volume.

Comments

WARNING! This function invalidates directory enumeration. You cannot continue enumerating files after renaming one of them with this function. If you need to operate on additional files in the directory, you must first restart the enumeration.

Example

Below is an example of how to use `VFSFileRename`. Note that the renamed file remains in the `/PALM/Programs` directory; `VFSFileRename` can't be used to move files from one directory to another.

```
// volRefNum must have been previously defined; most likely,
// it was returned by VFSVolumeEnumerate

err = VFSFileRename(volRefNum, "/PALM/Programs/foo.prc”,
   "bar.prc");
if (err != errNone) {
      // handle error...
}
```

Compatibility

Implemented only if the VFS Manager Feature Set is present.

VFSFileResize Function

Purpose

Change the size of an open file. This function only operates on files and cannot be used with directories.

Declared In

`VfsMgr.h`

Prototype

```
Err VFSFileResize (FileRef fileRef,
    UInt32 newSize)
```

Parameters

→ *`fileRef`*
: File reference returned from `VFSFileOpen()`.

→ *newSize*
The desired new size of the file. This can be larger or smaller then the current file size.

Returns Returns one of the following error codes:

`errNone`
No error.

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`vfsErrFileBadRef`
The specified file reference is invalid.

`vfsErrIsADirectory`
The specified file reference points to a directory instead of a file. This is an invalid operation on a directory.

`vfsErrNoFileSystem`
The VFS Manager cannot find an appropriate file system to handle the request.

`vfsErrVolumeFull`
There is not enough space left on the volume.

Comments The location of the file pointer is undefined after a call to this function.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSFileSize()

VFSFileSeek Function

Purpose Set the position within an open file from which to read or write. This function only operates on files and cannot be used with directories.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileSeek (FileRef fileRef,
   FileOrigin origin, Int32 offset)
```

Parameters → *fileRef*
File reference returned from `VFSFileOpen()`.

→ *origin*
: Origin to use when calculating the new position. The `offset` parameter indicates the desired new position relative to this origin, which can be one of the following:

`vfsOriginBeginning`
: The beginning of the file.

`vfsOriginCurrent`
: The current position within the file.

`vfsOriginEnd`
: The end of the file. Only negative offsets are allowed when `origin` is set to `vfsOriginEnd`.

→ *offset*
: Offset, either positive or negative, from the origin to which the current position should be set. A value of zero (0) positions you at the specified origin.

Returns Returns one of the following error codes:

`errNone`
: No error.

`expErrNotOpen`
: The file system library necessary for this call has not been installed or has not been opened.

`vfsErrFileBadRef`
: The specified file reference is invalid.

`vfsErrFileEOF`
: The file pointer is at the end of file.

`vfsErrIsADirectory`
: The specified file reference points to a directory instead of a file. This is an invalid operation on a directory.

`sysErrParamErr`
: The specified origin is not one of the defined constants.

Comments During a call to this function, if the resulting position would be beyond the end of the file, it sets the position to the end of the file.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSFileSize(), VFSFileTell()

VFSFileSetAttributes Function

Purpose Change the attributes of an open file or directory.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileSetAttributes (FileRef fileRef,
    UInt32 attributes)
```

Parameters

→ *`fileRef`*
File reference returned from `VFSFileOpen()`.

→ *`attributes`*
Attributes to associate with the file or directory. See File and Directory Attributes for a list of values you can use when setting this parameter:

Returns Returns one of the following error codes:

`errNone`
No error.

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`sysErrParamErr`
One of the parameters is invalid.

`vfsErrFileBadRef`
The specified file reference is invalid.

`vfsErrNoFileSystem`
The VFS Manager cannot find an appropriate file system to handle the request.

Comments

NOTE: You cannot use this function to set the `vfsFileAttrDirectory` or `vfsFileAttrVolumeLabel` attributes. The `vfsFileAttrDirectory` is set when you call `VFSDirCreate()`. The `vfsFileAttrVolumeLabel` is set when you call `VFSVolumeSetLabel()`. This function may fail when setting other attributes, depending on the underlying file system.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSFileGetAttributes(), VFSFileSetDate()

VFSFileSetDate Function

Purpose Changes the dates on an open file or directory.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileSetDate (FileRef fileRef,
    UInt16 whichDate, UInt32 date)
```

Parameters

→ *`fileRef`*
File reference returned in `VFSFileOpen()`.

→ *`whichDate`*
Specifies which date—creation, modification, or last access—to modify. Supply one of the following values:

`vfsFileDateCreated`

`vfsFileDateModified`

`vfsFileDateAccessed`
Note that not all file systems are required to support the above dates. If the supplied date type is not supported by the file system, `VFSFileGetDate` returns `expErrUnsupportedOperation`.

→ *`date`*
The new date. This field should be expressed in the standard Palm OS date format — the number of seconds since midnight (00:00:00) January 1, 1904.

Returns Returns one of the following error codes:

`errNone`
No error.

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`expErrUnsupportedOperation`
The specified date type is not supported by the underlying file system.

`sysErrParamErr`
The `whichDate` parameter is not one of the defined constants.

`vfsErrFileBadRef`
The specified file reference is invalid.

`vfsErrFilePermissionDenied`
: Write permission is not enabled for this file.

`vfsErrNoFileSystem`
: The VFS Manager cannot find an appropriate file system to handle the request.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSFileGetDate(), VFSFileSetAttributes()

VFSFileSize Function

Purpose Obtain the size of an open file. This function only operates on files and cannot be used with directories.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileSize (FileRef fileRef,
   UInt32 *fileSizeP)
```

Parameters

→ *fileRef*
: File reference returned from `VFSFileOpen()`.

← *fileSizeP*
: Pointer to the size of the open file.

Returns Returns one of the following error codes:

`errNone`
: No error.

`expErrNotOpen`
: The file system library necessary for this call has not been installed or has not been opened.

`vfsErrFileBadRef`
: The specified file reference is invalid.

`vfsErrIsADirectory`
: The specified file reference points to a directory instead of a file. This is an invalid operation on a directory.

`vfsErrNoFileSystem`
: The VFS Manager cannot find an appropriate file system to handle the request.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSFileResize(), VFSFileTell(), VFSVolumeSize()

VFSFileTell Function

Purpose Get the current position of the file pointer within an open file. This function only operates on files and cannot be used with directories.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileTell (FileRef fileRef,
    UInt32 *filePosP)
```

Parameters

→ *fileRef*
File reference returned from `VFSFileOpen()`.

← *filePosP*
Pointer to the current file position.

Returns Returns one of the following error codes:

`errNone`
No error.

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`vfsErrFileBadRef`
The specified file reference is invalid.

`vfsErrIsADirectory`
The specified file reference points to a directory instead of a file. This is an invalid operation on a directory.

`vfsErrNoFileSystem`
The VFS Manager cannot find an appropriate file system to handle the request.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSFileSeek(), VFSFileSize()

VFSFileWrite Function

Purpose Write data to an open file. This function only operates on files and cannot be used with directories.

Declared In `VfsMgr.h`

Prototype

```
Err VFSFileWrite (FileRef fileRef,
   UInt32 numBytes, const void *dataP,
   UInt32 *numBytesWrittenP)
```

Parameters

→ *fileRef*
File reference returned from `VFSFileOpen()`.

→ *numBytes*
The number of bytes to write.

→ *dataP*
Pointer to the data that is to be written.

← *numBytesWrittenP*
Pointer to an unsigned integer that reflects the number of bytes actually written. This value is set on return and does not need to be initialized. If no bytes are written the value is set to zero. Pass `NULL` for this parameter if you do not need to know how many bytes were written.

Returns Returns one of the following error codes:

`errNone`
No error.

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`vfsErrFileBadRef`
The specified file reference is invalid.

`vfsErrFilePermissionDenied`
Write permission is not enabled for this file.

`vfsErrIsADirectory`
The specified file reference points to a directory instead of a file. This is an invalid operation on a directory.

`vfsErrNoFileSystem`
The VFS Manager cannot find an appropriate file system to handle the request.

`vfsErrVolumeFull`
There is not enough space left on the volume.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSExportDatabaseToFile(), VFSExportDatabaseToFileCustom(), VFSFileRead(), VFSFileReadData()

VFSGetDefaultDirectory Function

Purpose Determine the default location on the given volume for files of a particular type.

Declared In `VfsMgr.h`

Prototype

```
Err VFSGetDefaultDirectory (UInt16 volRefNum,
   const Char *fileTypeStr, Char *pathStr,
   UInt16 *bufLenP)
```

Parameters

→ *volRefNum*
Volume reference number returned from `VFSVolumeEnumerate()`.

→ *fileTypeStr*
Pointer to the requested file type, as a null-terminated string. The file type may either be a MIME media type/subtype pair, such as "image/jpeg", “text/plain”, or “audio/basic”; or a file extension, such as “`.jpeg`.”

← *pathStr*
Pointer to the buffer which receives the default directory path for the requested file type.

↔ *bufLenP*
Pointer to the size of the path. Set this to the size of `pathStr` buffer on input. Reflects the number of bytes copied to `pathStr` on output.

Returns Returns one of the following error codes:

`errNone`
No error.

`vfsErrBadName`
There is no default directory registered for the requested file type.

`vfsErrBufferOverflow`
: A match was found, but the `pathStr` buffer is too small to hold the resulting path string. A partial path is returned in `pathStr`.

`vfsErrFileNotFound`
: No match was found for the specified volume. The error could have occurred with either the media type specified for this volume or the file type requested.

Comments

This function returns the complete path to the default directory registered for the specified file type. A default directory can be registered for each type of media supported. The directory should be registered under media and file type. Note that this directory is typically a "root" directory for the file type; any subdirectories under this root directory should also be searched for files of the appropriate type.

This function can be used by an image viewer application, for example, to find the directory containing images without having to know what type of media the volume was on. This could be "/DCIM", "/images", or something else depending on the type of media.

Compatibility

Implemented only if the VFS Manager Feature Set is present.

See Also

VFSDirEntryEnumerate(), VFSRegisterDefaultDirectory(), VFSUnregisterDefaultDirectory()

VFSImportDatabaseFromFile Function

Purpose

Creates a database from a `.pdb` or `.prc` file on an external storage card.

Declared In

`VfsMgr.h`

Prototype

```
Err VFSImportDatabaseFromFile (UInt16 volRefNum,
    const Char *pathNameP, UInt16 *cardNoP,
    LocalID *dbIDP)
```

Parameters

→ *`volRefNum`*
: Volume on which the source file resides.

→ *`pathNameP`*
: Pointer to the full path and name of the source file.

← *cardNoP*
: Pointer to a variable that receives the card number of the newly-created database. If the database already resides in the storage heap, the card number of the existing database is returned along with the error `dmErrAlreadyExists`.

← *dbIDP*
: Pointer to a variable that receives the database ID of the new database. If the database already resides in the storage heap, the database ID of the existing database is returned along with the error `dmErrAlreadyExists`.

Returns Returns one of the following error codes:

`errNone`
: No error.

`dmErrAlreadyExists`
: The `.prc` or `.pdb` file already exists in the storage heap. In this case the `cardNoP` and `dbIDP` are set to point to the existing file.

`expErrNotEnoughPower`
: There is insufficient battery power to complete the requested operation.

`vfsErrBadName`
: The path name specified in `pathNameP` is not valid.

Comments This utility function imports a `.pdb` or `.prc` file resident on an external storage card into a new database in the storage heap. It first calls `VFSFileOpen()` to open the file specified in `pathNameP`. Assuming that a corresponding `.prc` or `.pdb` does not already exist in the storage heap, `VFSImportDatabaseFromFile()` calls the Exchange Manager function `ExgDBRead()` with an internal callback function for importing a file to the Data Manager. The Exchange Manager makes repeated calls to this function, which passes the data back in blocks. Once the file has been successfully imported, the owner (the imported file, if it's an executable, or the associated application if it is not) is sent a `sysAppLaunchCmdSyncNotify` launch code to make it aware of the new database.

This function only imports the specified `.pdb` or `.prc` file; it does not import bundled databases or overlays. If there are bundled databases and/or overlays associated with the `.pdb` or `.prc` file

you are importing, you will need to write additional code to explicitly handle them.

This function doesn't provide any progress indication to the user. If you need to provide feedback to the user as the file import progresses, use `VFSImportDatabaseFromFileCustom()` instead.

This function is used, for example, to copy applications from a storage card to main memory.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSExportDatabaseToFile(), VFSFileRead()

VFSImportDatabaseFromFileCustom Function

Purpose Create a database from the specified .pdb or `.prc` file on an external storage card. This function differs from `VFSImportDatabaseFromFile()` in that it allows you to track the progress of the import operation.

Declared In `VfsMgr.h`

Prototype

```
Err VFSImportDatabaseFromFileCustom
    (UInt16 volRefNum, const Char *pathNameP,
    UInt16 *cardNoP, LocalID *dbIDP,
    VFSImportProcPtr importProcP, void *userDataP)
```

Parameters

→ *volRefNum*
Volume on which the source file resides.

→ *pathNameP*
Pointer to the full path and name of the source file.

← *cardNoP*
Pointer to the variable that receives the card number of the newly-created database. If the database already resides in the storage heap, the card number of the existing database is returned along with the error `dmErrAlreadyExists`.

← *dbIDP*
Pointer to the variable that receives the database ID of the new database. If the database already resides in the storage heap, the database ID of the existing database is returned along with the error `dmErrAlreadyExists`.

→ *importProcP*
User-defined callback function that tracks the progress of the import. This function should allow the user to cancel the import. Pass `NULL` if you don't have a progress callback function. See `VFSImportProcPtr()` for the requirements of this function.

→ *userDataP*
Pointer to any data you want to pass to the callback function specified in `importProcP`. This information is not used internally by the VFS Manager. Pass `NULL` if you don't have a progress callback function, or if that function doesn't need any such data.

Returns Returns one of the following error codes:

`errNone`
No error

`vfsErrBadName`
The path name specified in `pathNameP` is not valid.

`expErrNotEnoughPower`
The power required to import a database is not available.

`dmErrAlreadyExists`
The `.prc` or `.pdb` file already exists in main memory. In this case the `cardNoP` and `dbIDP` are set to point to the existing file.

Comments This function is similar to `VFSImportDatabaseFromFile()` in that it imports a `.pdb` or `.prc` file on an external storage card into a new database on the storage heap. It extends the functionality by allowing you to specify a callback function that tracks the progress of the export. It first calls `VFSFileOpen()` to open the file specified in `pathNameP`. If a corresponding `.prc` or `.pdb` does not already exist in main memory, it calls the Exchange Manager function `ExgDBRead()` with an internal callback function for importing the file from the Data Manager. The Exchange Manager makes repeated calls to this function, which receives the data back in blocks. The progress tracker, if one has been specified, is also called every time a new chunk of data is passed back. Once the file has been successfully imported, the owner (the imported file, if it's an executable, or the associated application if it is not) is sent a

sysAppLaunchCmdSyncNotify launch code to make it aware of the new database.

Like VFSImportDatabaseFromFile, this function only imports the specified .pdb or .prc file; it does not import bundled databases or overlays.

This function is used, for example, to copy applications from a storage card to main memory.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSFileRead(), VFSExportDatabaseToFileCustom()

VFSInstallFSLib Function

Purpose Install a file system library so that the VFS Manager can use it.

Declared In VfsMgr.h

Prototype

```
Err VFSInstallFSLib (UInt32 creator,
   UInt16 *fsLibRefNumP)
```

Parameters

→ *creator*
: Creator ID of the database containing the file system library to be installed.

← *fsLibRefNumP*
: Pointer to the reference number for the newly installed file system library. Supply NULL for this parameter if you don't need the library reference number.

Returns If the file system library was loaded and installed without error, errNone is returned. Any error generated by the underlying file system while opening the file system library or determining its type will be returned from VFSInstallFSLib. Other errors that can be generated during the file system library installation process include:

expErrIncompatibleAPIVer
: The file system library has an incompatible API version.

expErrNotOpen
: The file system library necessary for this call has not been installed or has not been opened.

`memErrNotEnoughSpace, memErrChunkNotLocked, or memErrChunkLocked`
: A memory problem occurred while inserting the library reference into the list of installed libraries.

`sysErrLibNotFound, sysErrNoFreeRAM, sysErrNoFreeLibSlots (or some other error returned from the library's install entry point)`
: An error occurred while loading the library.

Comments This function calls `SysLibLoad()` to load the file system library into the library table. Once loaded the appropriate file system is asked to open the library. At boot time `VFSInstallFSLib` is called internally by the Expansion Manager to load all installed file system libraries and initialize them for use.

`VFSInstallFSLib` is not normally called by applications.

Compatibility Implemented only if the VFS Manager Feature Set is present. This function is not implemented, however, if Palm OS Cobalt Feature Set is present.

See Also VFSRemoveFSLib()

VFSRegisterDefaultDirectory Function

Purpose Registers a specific directory as the default location for files of a given type on a particular kind of external storage card. This function is generally called by a slot driver for files and media types that are supported by that slot driver.

Declared In `VfsMgr.h`

Prototype

```
Err VFSRegisterDefaultDirectory
    (const Char *fileTypeStr, UInt32 mediaType,
    const Char *pathStr)
```

Parameters → *`fileTypeStr`*
: Pointer to the file type to register. This is a null-terminated string that can either be a MIME media type/subtype pair, such as "image/jpeg", "text/plain", or "audio/basic"; or a file extension, such as "`.jpeg`".

→ *mediaType*
: Type of card media for which the default directory is being associated. See Defined Media Types in the Expansion Manager chapter for the list of accepted values.

→ *pathStr*
: Pointer to the default directory path to be associated with the specified file type. This string must be null-terminated, and must be the full path to the directory.

Returns Returns one of the following error codes:

`errNone`
: No error.

`sysErrParamErr`
: Either the `fileTypeStr` parameter is `NULL` or the `pathStr` parameter is `NULL`.

`vfsErrFileAlreadyExists`
: A default directory has already been registered for this file type on the specified card media type.

Comments This function first verifies that a default directory has not already been registered for the specified combination of file type and media type, and returns `vfsErrFileAlreadyExists` if one has been registered. To change an existing entry in the registry, you must first remove the existing entry with a call to `VFSUnregisterDefaultDirectory()` before re-registering it with `VFSRegisterDefaultDirectory`.

The specified directory registered for a given file type is intended to be the "root" default directory. If a given default directory has one or more subdirectories, applications should also search those subdirectories for files of the appropriate type.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSGetDefaultDirectory()

VFSRemoveFSLib Function

Purpose Remove a file system library from the library table, so that the VFS Manager can no longer use it.

Declared In `VfsMgr.h`

Prototype

```
Err VFSRemoveFSLib (UInt16 fsLibRefNum)
```

Parameters → *`fsLibRefNum`*
Library reference number of the file system library to remove from the library table.

Returns Returns one of the following error codes:

`errNone`
No error.

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`vfsErrNoFileSystem`
VFS Manager can not find the file system specified in `fsLibRefNum`.

Comments This function is not normally called by applications. It unmounts any volumes that the specified file system may have mounted. It then closes the library and removes it from the library table with `SysLibRemove()`.

Compatibility Implemented only if the VFS Manager Feature Set is present. This function is not implemented, however, if Palm OS Cobalt Feature Set is present.

VFSUnregisterDefaultDirectory Function

Purpose Sever the association between a particular file type and a default directory for a given type of card media.

Declared In `VfsMgr.h`

Prototype

```
Err VFSUnregisterDefaultDirectory
    (const Char *fileTypeStr, UInt32 mediaType)
```

Parameters → *`fileTypeStr`*
Pointer to the file type with which the default directory is associated. This is a null-terminated string that can either be

a MIME media type/subtype pair, such as "image/jpeg", "text/plain", or "audio/basic"; or a file extension, such as "`.jpeg`".

→ *`mediaType`*
Type of card media for which the default directory is associated. See Defined Media Types in the Expansion Manager chapter for the list of accepted values.

Returns Returns one of the following error codes:

`errNone`
No error.

`sysErrParamErr`
The `fileTypeStr` parameter is `NULL`.

`vfsErrFileNotFound`
A default directory could not be found in the registry for the specified file and media type.

Comments

NOTE: Caution is advised when using this function, since you may remove another application's registration, causing data to mysteriously disappear from those applications.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSGetDefaultDirectory(), VFSRegisterDefaultDirectory()

VFSVolumeEnumerate Function

Purpose Enumerate the mounted volumes.

Declared In `VfsMgr.h`

Prototype

```
Err VFSVolumeEnumerate (UInt16 *volRefNumP,
    UInt32 *volIteratorP)
```

Parameters

← *`volRefNumP`*
Pointer to the reference number for the volume represented by the current enumeration, or `vfsInvalidVolRef` if there are no more volumes to be enumerated or an error occurred.

↔ *`volIteratorP`*
Pointer to a variable that holds the index of the current enumeration. Set the variable to `vfsIteratorStart` prior to the first iteration. Each call to `VFSVolumeEnumerate`

updates the variable to the index of the next volume. When the last volume is reached, the variable pointed to by `volIteratorP` is set to `vfsIteratorStop`.

Returns Returns one of the following error codes:

`errNone`
: No error

`expErrEnumerationEmpty`
: There are no volumes to enumerate.

`sysErrParamErr`
: The value pointed to by `volIteratorP` is not valid. This error is also returned when `volIteratorP` is `vfsIteratorStop`.

Comments This function returns a pointer to the volume reference number in the `volRefNumP` parameter. In order to traverse all volumes you must make repeated calls to `VFSVolumeEnumerate()` inside a loop. Before the first call to `VFSVolumeEnumerate`, the variable pointed to by `volIteratorP` should be initialized to `vfsIteratorStart`. Each iteration then increments `volIteratorP` to the next entry after updating `volRefNumP`. When the last volume is reached, `*volIteratorP` is set to `vfsIteratorStop`. If there are no volumes to enumerate, `VFSVolumeEnumerate` returns `expErrEnumerationEmpty` when first called.

Example Below is an example of how to use `VFSVolumeEnumerate`.

```
UInt16 volRefNum;
UInt32 volIterator = vfsIteratorStart;

while (volIterator != vfsIteratorStop) {
   err = VFSVolumeEnumerate(&volRefNum, &volIterator);
   if (err == errNone) {
      // Do something with the volRefNum
   } else {
      // handle error... possibly by
      // breaking out of the loop
   }
}
```

Compatibility Implemented only if the VFS Manager Feature Set is present.

VFSVolumeFormat Function

Purpose Format and mount the volume installed in a given slot.

Declared In `VfsMgr.h`

Prototype

```
Err VFSVolumeFormat (UInt8 flags,
    UInt16 fsLibRefNum,
    VFSAnyMountParamPtr vfsMountParamP)
```

Parameters

→ *flags*
: Flags that control how the volume should be formatted. Currently, the only flag not reserved is `vfsMountFlagsUseThisFileSystem`. Pass this flag to cause the volume to be formatted using the file system specified by `fsLibRefNum`. Pass zero (0) to have the VFS Manager attempt to format the volume using a file system appropriate to the slot.

→ *fsLibRefNum*
: Reference number of the file system library for which the volume should be formatted. This number is obtained through a call to `SysLibFind()` with the name of the library you want to use. If the `flags` field is not set to `vfsMountFlagsUseThisFileSystem`, this parameter is ignored.

↔ *vfsMountParamP*
: Parameters to be used when formatting the volume and when mounting the volume after it has been formatted. Supply a pointer to either a `VFSSlotMountParamType` or a `VFSPOSEMountParamType` structure. Note that you'll need to cast your structure pointer to a `VFSAnyMountParamPtr`. Set the `mountClass` field to the appropriate value: if you are mounting to an Expansion Manager slot, set `mountClass` to `VFSMountClass_SlotDriver` and initialize `slotLibRefNum` and `slotRefNum` to the appropriate values. See the descriptions of `VFSAnyMountParamType`, `VFSSlotMountParamType`, and `VFSPOSEMountParamType` for information on the fields that make up these data structures.

Returns Returns one of the following error codes:

`errNone`
: No error.

`expErrNotEnoughPower`
: There is insufficient battery power to format and/or mount a volume.

`expErrNotOpen`
: The file system library necessary for this call has not been installed or has not been opened.

`vfsErrNoFileSystem`
: The VFS Manager cannot find an appropriate file system to handle the request.

Comments

The slot driver currently only supports one volume per slot. If the volume is successfully formatted and mounted, the reference number of the mounted volume is returned in `vfsMountParamP->volRefNum`. If the format is unsuccessful or cancelled, `vfsMountParamP->volRefNum` is set to `vfsInvalidVolRef`.

If `vfsMountFlagsUseThisFileSystem` is passed as a flag, `VFSVolumeFormat` attempts to format the volume using the file system library specified by `fsLibRefNum`. Typically the flag parameter is not set. In this case `VFSVolumeFormat` tries to find a compatible library to format the volume, as follows:

1. Check to see if the default file system library feature is set. If it is, and if that file system is installed, it is used to format the volume. You can set the default file system using `FtrSet()`; supply `sysFileCVFSMgr` for the feature creator, and `vfsFtrIDDefaultFS` for the feature number.
2. Check to see if any of the installed file systems are natively supported for the slot on which the VFS Manager is trying to format. If one of them is, it is used to format the volume.
3. If none of the installed file systems can perform the format using the slot's native type, a dialog displays warning the user that their media may become incompatible with other devices if they continue with the format. The user may continue or cancel the format. If the user chooses to continue, `VFSVolumeFormat` formats the volume using the first file system library that was installed.

When calling `VFSVolumeFormat`, the volume can either be mounted or unmounted. The underlying file system library call requires the volume to be unmounted. `VFSVolumeFormat` checks to see if the volume is currently mounted and unmounts it, if

necessary, using VFSVolumeUnmount before making the file system call. If the file system successfully formats the volume, VFSVolumeFormat mounts it and posts a sysNotifyVolumeMountedEvent notification.

Example The following code excerpt formats a volume on an Expansion Manager slot using a compatible file system.

```
VFSSlotMountParamType slotParam;
UInt32 slotIterator = expIteratorStart;

slotParam.vfsMountParamP.mountClass =
   VFSMountClass_SlotDriver;
err = ExpSlotEnumerate(&slotParam.slotRefNum,
   &slotIterator);
err = ExpSlotLibFind(slotParam.slotRefNum,
   &slotParam.slotLibRefNum);

err = VFSVolumeFormat(NULL, NULL,
   (VFSAnyMountParamPtr) & slotParam);
```

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSVolumeMount()

VFSVolumeGetLabel Function

Purpose Determine the volume label for a particular volume.

Declared In VfsMgr.h

Prototype

```
Err VFSVolumeGetLabel(UInt16 volRefNum,
    Char *labelP, UInt16 bufLen)
```

Parameters

→ *volRefNum*
Volume reference number returned from VFSVolumeEnumerate().

← *labelP*
Pointer to a character buffer into which the volume name is placed.

→ *bufLen*
Length, in bytes, of the labelP buffer.

Returns Returns one of the following error codes:

`errNone`
No error.

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`vfsErrNoFileSystem`
The VFS Manager cannot find an appropriate file system to handle the request.

`vfsErrVolumeBadRef`
The specified volume has not been mounted.

`vfsErrBufferOverflow`
The value specified in `bufLen` is not big enough to receive the full volume label.

`vfsErrNameShortened`
There was an error reading the full volume name. A shortened version is being returned.

Comments Volume reference numbers can change each time you mount a given volume. To keep track of a particular volume, save the volume's label rather than its reference number. Volume labels can be up to 255 characters long. They can contain any normal character, including spaces and lower case characters, in any character set as well as the following special characters: $ % ' - _ @ ~ ` ! ( ) ^ # & + , ; = [ ].

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSVolumeSetLabel()

VFSVolumeInfo Function

Purpose Get information about the specified volume.

Declared In `VfsMgr.h`

Prototype

```
Err VFSVolumeInfo(UInt16 volRefNum, VolumeInfoType
    *volInfoP)
```

Parameters → *`volRefNum`*
Volume reference number returned from `VFSVolumeEnumerate()`.

← *volInfoP*
Pointer to the structure that receives the volume information for the specified volume. See `VolumeInfoType` for more information on the fields in this data structure.

Returns Returns one of the following error codes:

`errNone`
No error.

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`vfsErrNoFileSystem`
The VFS Manager cannot find an appropriate file system to handle the request.

`vfsErrVolumeBadRef`
The specified volume reference number is invalid.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSVolumeGetLabel(), VFSVolumeSize()

VFSVolumeMount Function

Purpose Mount the card's volume on the specified slot.

Declared In `VfsMgr.h`

Prototype

```
Err VFSVolumeMount(UInt8 flags,
   UInt16 fsLibRefNum,
   VFSAnyMountParamPtr vfsMountParamP)
```

Parameters

→ *flags*
Flags that control how the volume should be mounted. Currently, the only flag not reserved is `vfsMountFlagsUseThisFileSystem`. Pass this flag to cause the volume to be mounted using the file system specified by `fsLibRefNum`. Pass zero (0) to have the VFS Manager attempt to mount the volume using a file system appropriate for the slot.

→ *fsLibRefNum*
Reference number of the file system library for which the volume should be mounted. This number is obtained

through a call to SysLibFind() with the name of the library you want to use. If the flags field is not set to vfsMountFlagsUseThisFileSystem, this parameter is ignored.

↔ *vfsMountParamP*
: Parameters to be used when mounting the volume after it has been formatted. Supply a pointer to either a VFSSlotMountParamType or a VFSPOSEMountParamType structure. Note that you'll need to cast your structure pointer to a VFSAnyMountParamPtr. Set the mountClass field to the appropriate value: if you are mounting to an Expansion Manager slot, set mountClass to VFSMountClass_SlotDriver and initialize slotLibRefNum and slotRefNum to the appropriate values. See the descriptions of VFSAnyMountParamType, VFSSlotMountParamType, and VFSPOSEMountParamType for information on the fields that make up these data structures.

Returns Returns one of the following error codes:

errNone
: No error.

expErrNotEnoughPower
: There is insufficient battery power to mount a volume.

expErrNotOpen
: The file system library necessary for this call has not been installed or has not been opened.

sysErrParamErr
: vfsMountParamP was initialized to NULL.

vfsErrNoFileSystem
: The VFS Manager cannot find an appropriate file system to handle the request.

vfsErrVolumeStillMounted
: The volume is already mounted with a different file system than was specified in fsLibRefNum.

Comments The slot driver only supports one volume per slot. The reference number of the mounted volume is returned in vfsMountParamP->volRefNum. If vfsMountFlagsUseThisFileSystem is passed as a flag, VFSVolumeMount attempts to mount the volume using

the file system library specified by `fsLibRefNum`. Otherwise `VFSVolumeMount` tries to find a file system library which is able to mount the volume. If none of the installed file system libraries is able to mount the volume, `VFSVolumeMount` attempts to re-format the volume (using `VFSVolumeFormat()`) and then mount it. If `VFSVolumeMount` manages to successfully mount the volume, it ends by posting a `sysNotifyVolumeMountedEvent` notification.

After VFSVolumeMount successfully mounts a volume, it broadcasts `sysNotifyVolumeMountedEvent`. The VFS Manager, upon being notified of this event, searches the newly-mounted volume for `/PALM/start.prc`. If `start.prc` is found in the `/PALM` directory, the VFS Manager copies it to main memory and launches it. If start.prc is not found, the VFS Manager switches to the Launcher instead. This behavior can be overridden; see Card Insertion and Removal in Chapter 7, "Expansion," *Palm OS Programmer's Companion*, vol. I.

When `VFSVolumeMount` is called, if the volume is already mounted with a different file system than was specified in `fsLibRefNum`, a `vfsErrVolumeStillMounted` error is returned. If the volume is already mounted with the same file system that is specified in `fsLibRefNum`, or if `vfsMountFlagsUseThisFileSystem` is not set, `VFSVolumeMount` returns `errNone` and sets `volRefNumP` to the reference number of the currently mounted volume.

Example

The following code excerpt mounts a volume on an Expansion Manager slot using a compatible file system.

```
VFSSlotMountParamType slotParam ;
UInt32 slotIterator = expIteratorStart;

slotParam.vfsMountParamP.mountClass =
   VFSMountClass_SlotDriver;
err = ExpSlotEnumerate(&slotParam.slotRefNum,
   &slotIterator);
err = ExpSlotLibFind(slotParam.slotRefNum,
   &slotParam.slotLibRefNum);

err = VFSVolumeMount(NULL, NULL,
   (VFSAnyMountParamPtr) & slotParam);
```

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSVolumeFormat(), VFSVolumeUnmount()

VFSVolumeSetLabel Function

Purpose Changes the volume label for a mounted volume.

Declared In `VfsMgr.h`

Prototype

```
Err VFSVolumeSetLabel(UInt16 volRefNum,
   const Char *labelP)
```

Parameters

→ *volRefNum*
Volume reference number returned from `VFSVolumeEnumerate()`.

→ *labelP*
Pointer to the label to be applied to the specified volume. This string must be null-terminated.

Returns Returns one of the following error codes:

`errNone`
No error

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`vfsErrBadName`
The supplied label is invalid.

`vfsErrNameShortened`
Indicates that the label name was too long. A shortened version of the label name was used instead.

`vfsErrVolumeBadRef`
The specified volume has not been mounted.

Comments Volume labels can be up to 255 characters long. They can contain any normal character, including spaces and lower case characters, in any character set as well as the following special characters: $ % ' - _ @ ~ ` ! ( ) ^ # & + , ; = [ ]. See Naming Volumes in Chapter 7, "Expansion,"*Palm OS Programmer's Companion*, vol. I for guidelines on naming.

NOTE: Most clients should not need to call this function. This function may create or delete a file in the root directory, which would invalidate any current calls to `VFSDirEntryEnumerate()`.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSVolumeGetLabel()

VFSVolumeSize Function

Purpose Determine the total amount of space on a volume, as well as the amount that is currently being used.

Declared In `VfsMgr.h`

Prototype

```
Err VFSVolumeSize(UInt16 volRefNum,
   UInt32 *volumeUsedP, UInt32 *volumeTotalP)
```

Parameters

→ *volRefNum*
Volume reference number returned from `VFSVolumeEnumerate()`.

← *volumeUsedP*
Pointer to a variable that receives the amount of space, in bytes, in use on the volume.

← *volumeTotalP*
Pointer to a variable that receives the total amount of space on the volume, in bytes.

Returns Returns one of the following error codes:

`errNone`
No error.

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`vfsErrNoFileSystem`
The VFS Manager cannot find an appropriate file system to handle the request.

`vfsErrVolumeBadRef`
The specified volume has not been mounted.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also `VFSVolumeInfo()`

VFSVolumeUnmount Function

Purpose Unmount the given volume.

Declared In `VfsMgr.h`

Prototype

```
Err VFSVolumeUnmount(UInt16 volRefNum)
```

Parameters

→ *`volRefNum`*
Volume reference number returned from `VFSVolumeEnumerate()`.

Returns Returns one of the following error codes:

`errNone`
No error.

`expErrNotOpen`
The file system library necessary for this call has not been installed or has not been opened.

`vfsErrNoFileSystem`
The VFS Manager cannot find an appropriate file system to handle the request.

`vfsErrVolumeBadRef`
The specified volume has not been mounted.

Comments This function closes any opened files and posts a `sysNotifyVolumeUnmountedEvent` notification once the file system is successfully unmounted.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also VFSVolumeMount()

Application-Defined Functions

VFSExportProcPtr Function

Purpose User-defined callback function supplied to `VFSExportDatabaseToFileCustom()` that tracks the progress of the export.

Declared In `VfsMgr.h`

Prototype

```
typedef Err (*VFSExportProcPtr)
   (UInt32 totalBytes, UInt32 offset,
   void *userDataP)
```

Parameters

`totalBytes`
: The total number of bytes being exported.

`offset`
: Undefined.

`userDataP`
: Pointer to any application-specific data passed to the callback function. This pointer may be `NULL` if your callback doesn't need any such data.

Returns Your progress tracker should allow the user to abort the export. Return `errNone` if the export should continue, or any other value to abort the export process. If you return a value other than `errNone`, that value will be returned by `VFSExportDatabaseToFileCustom()`.

Comments See the Progress Dialogs section in the *Palm OS Programmer's Companion*, vol. I for more information on writing a progress tracker.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also `VFSImportProcPtr()`

VFSImportProcPtr Function

Purpose User-defined callback function supplied to `VFSImportDatabaseFromFileCustom()` that tracks the progress of the import.

Declared In `VfsMgr.h`

Prototype

```
typedef Err (*VFSImportProcPtr)
    (UInt32 totalBytes, UInt32 offset,
    void *userDataP)
```

Parameters

`totalBytes`
The total number of bytes being imported.

`offset`
The number of bytes that have already been imported. This value, along with the total number of bytes being imported, allows you to inform the user how far along the import is.

`userDataP`
Pointer to NY application-specific data passed to the callback function. This pointer may be `NULL` if your callback doesn't need any such data.

Returns Your progress tracker should allow the user to abort the import. Return `errNone` if the import should continue, or any other value to abort the import process. If you return a value other than `errNone`, that value will be returned by `VFSImportDatabaseFromFileCustom()`.

Comments See the Progress Dialogs section in the *Palm OS Programmer's Companion*, vol. I for more information on writing a progress tracker.

Compatibility Implemented only if the VFS Manager Feature Set is present.

See Also `VFSExportProcPtr()`

59

Windows

This chapter provides information about windows by discussing these topics:

- Window Data Structures
- Window Constants
- Window Functions

No resources are associated with window objects.

The header file `Window.h` declares the API that this chapter describes. For more information on windows, see the section "Drawing on the Palm Powered Handheld" in the *Palm OS Programmer's Companion*, vol. I.

Window Data Structures

CustomPatternType Typedef

Purpose The `CustomPatternType` type holds an 8-by-8 bit pattern that is one bit deep. Each byte specifies a row of the pattern. When drawing, a pattern is tiled to fill a specified region. This pattern is used by `WinFillLine()` and `WinFillRectangle()`.

The `PatternType` specifies the name of the current pattern.

Prototype

```
typedef UInt8 CustomPatternType [8];
```

Compatibility In pre-3.5 systems, the `CustomPatternType` is an array of 4 16-bit words. Note the size of this data type has not changed.

DrawStateFlagsType Struct

Purpose The DrawStateFlagsType structure defines a set of flags that control how bitmaps and text are scaled. Use `WinSetScalingMode()` to

set these flags, and `WinGetScalingMode()` to obtain the values of these flags.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `DrawStateFlagsType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct DrawStateFlagsType {
  UInt16 unscaledBitmaps:1;
  UInt16 unscaledText:1;
  UInt16 unpaddedText:1;
  UInt16 reserved:13;
} DrawStateFlagsType;
```

Fields

`unscaledBitmaps`
: If set, low-density bitmaps are used, unscaled.

`unscaledText`
: If set, low density fonts are used, unscaled.

`unpaddedText`
: If set, padding pixels aren't inserted between text glyphs.

`reserved`
: Reserved for future use.

Compatibility Defined only if the 1.5x Display Feature Set is present.

DrawStateType Struct

Purpose The `DrawStateType` structure defines the current drawing state, which is the Palm OS® implementation of a **pen**. This drawing state is saved with `WinPushDrawState()` and restored with `WinPopDrawState()`.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `DrawStateType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct DrawStateType {
  WinDrawOperation transferMode;
  PatternType pattern;
  UnderlineModeType underlineMode;
  FontID fontId;
  FontPtr font;
  CustomPatternType patternData;
  IndexedColorType foreColor;
  IndexedColorType backColor;
  IndexedColorType textColor;
  UInt8 reserved;
  RGBColorType foreColorRGB;
  RGBColorType backColorRGB;
  RGBColorType textColorRGB;
  UInt16 coordinateSystem;
  DrawStateFlagsType flags;
  Fixed16 scale;
  Fixed16 ntvToActiveScale;
  Fixed16 stdToActiveScale;
  Fixed16 activeToStdScale;
} DrawStateType;
```

Fields

`transferMode`
: The current transfer mode for color drawing. See `WinDrawOperation`. Use `WinSetDrawMode()` to set this value.

`pattern`
: The name of the current pattern. See `PatternType`. If set to `customPattern`, the `patternData` field contains the actual pattern. Use `WinGetPatternType()` and `WinSetPatternType()` to retrieve and set this value.

`underlineMode`
: The current underline mode. See `UnderlineModeType`. Use `WinSetUnderlineMode()` to set this value.

`fontId`
: The ID of the current font. Use `FntSetFont()` to set this value.

`font`
: A pointer to the current font. Use `FntSetFont()` to set this value.

`patternData`
: The current pattern being used by the `WinFill` functions if `pattern` is `customPattern`. See `CustomPatternType`. Use `WinGetPattern()` and `WinSetPattern()` to retrieve and set this value.

`foreColor`
: Index of the current color used for the foreground. Use `WinSetForeColor()` to set this value. This field is only valid for indexed color bitmaps.

`backColor`
: Index of the current color used for the background. Use `WinSetBackColor()` to set this value. This field is only valid for indexed color bitmaps.

`textColor`
: Index of the current color used for text. Use `WinSetTextColor()` to set this value. This field is only valid for indexed color bitmaps.

`reserved`
: Reserved for future use.

`foreColorRGB`
: RGB value of the current color used for the foreground. Use `WinSetForeColorRGB()` to set this value. This field is only valid for Palm OS 4.0 or later, and only valid for direct color bitmaps.

`backColorRGB`
: RGB value of the current color used for the background. Use `WinSetBackColorRGB()` to set this value. This field is only valid for Palm OS 4.0 or later, and only valid for direct color bitmaps.

`textColorRGB`
: RGB value of the current color used for text. Use `WinSetTextColorRGB()` to set this value. This field is only valid for Palm OS 4.0 or later, and only valid for direct color bitmaps.

`coordinateSystem`
: Active coordinate system.

`flags`
: A `DrawStateFlagsType` structure, the contents of which control the scaling of bitmaps and text.

`scale`
: A fixed point value used to convert from the draw window's active coordinate system to native coordinates. This field is defined only if the High-Density Display Feature Set is present.

`ntvToActiveScale`
: A fixed point value used to convert from the native coordinate system to the draw window's active coordinate system. This field is defined only if the High-Density Display Feature Set is present.

`stdToActiveScale`
: A fixed point value used to convert from the standard coordinate system to the draw window's active coordinate system. This field is used internally to convert font metrics, which are stored as standard coordinates. This field is defined only if the High-Density Display Feature Set is present.

`activeToStdScale`
: The inverse of `stdToActive`; the active-to-standard scaling factor. This field is defined only if the High-Density Display Feature Set is present.

Compatibility

This type is implemented only if 3.5 New Feature Set is present. The `scale`, `ntvToActiveScale`, `stdToActiveScale`, and `activeToStdScale` fields are defined only if the High-Density Display Feature Set is present. The `flags` field is defined only if the 1.5x Display Feature Set is present; otherwise, the contents of `flags` are reserved.

FrameBitsType Struct

Purpose

The `FrameBitsType` structure specifies attributes of a window's frame.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `FrameBitsType` bit field. Never access its bit field members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef union FrameBitsType {
  struct {
    UInt16 cornerDiam  : 8;
    UInt16 reserved_3  : 3;
    UInt16 threeD      : 1;
    UInt16 shadowWidth : 2;
    UInt16 width       : 2;
  } bits;
  UInt16 word;
} FrameBitsType;
```

Fields

`cornerDiam`
: Corner radius of frame; maximum is 38.

`reserved_3`
: Reserved.

`threeD`
: Set this bit to draw a 3D button. This feature is not currently supported.

`shadowWidth`
: Width of shadow.

`width`
: Frame width.

`word`
: Reserved.

FrameType Typedef

Purpose The `FrameType` type specifies a window frame style.

Prototype

```
typedef UInt16 FrameType;
```

The `FrameType` can be set to one of the defined frame types shown in the table below, or a custom frame type as defined by a `FrameBitsType` structure.

Constant	Value	Description
`noFrame`	0	No frame
`simpleFrame`	1	Plain rectangular frame
`rectangleFrame`	1	Plain rectangular frame
`simple3DFrame`	0x0012	3D frame with width of 2. This frame type is not supported.
`roundFrame`	0x0401	Round frame with width of 1.
`boldRoundFrame`	0x0702	Round frame with width of 2.
`popupFrame`	0x0205	Popup frame style with slight corner roundness, width of 1 and shadow of 1.
`dialogFrame`	0x0302	Dialog frame style with slight corner roundness and width of 2.
`menuFrame`	`popupFrame`	Same as `popupFrame`.

IndexedColorType Typedef

Purpose The `IndexedColorType` type is used to specify a color by its index value; that is, by its location in a color table. Color tables are defined by the `ColorTableType` structure, which is declared in `Bitmap.h`. The `IndexedColorType` can hold a 1, 2, 4, or 8-bit index.

Prototype

```
typedef UInt8 IndexedColorType;
```

Compatibility This type is implemented only if 3.5 New Feature Set is present.

PatternType Enum

Purpose

The `PatternType` enumerated type specifies a pattern for drawing. This type is returned by `WinGetPatternType()` and is used as a parameter to the `WinSetPatternType()` function.

Prototype

```
typedef enum {
  blackPattern,
  whitePattern,
  grayPattern,
  customPattern,
  lightGrayPattern,
  darkGrayPattern
} PatternType;
```

Constants

`blackPattern`
: Pattern with all bits on.

`whitePattern`
: Pattern with all bits off.

`grayPattern`
: Pattern with alternating on and off bits.

`customPattern`
: Custom pattern specified by `CustomPatternType`.

`lightGrayPattern`
: Pattern with one out of four bits in each row turned on. This value is defined only if the High-Density Display Feature Set is present.

`darkGrayPattern`
: Pattern with one out of four bits in each row turned off. This value is defined only if the High-Density Display Feature Set is present.

These patterns all operate with current foreground and background color instead of black and white. In effect, `blackPattern` is only black if the current foreground color is black. `whitePattern` uses the current background color. `grayPattern` and `customPattern` uses a combination of background and foreground colors.

Patterns are expanded to the destination bit depth by the blitter when drawing patterned lines and filled rectangles.

The three standard gray patterns—`grayPattern`, `lightGrayPattern`, and `darkGrayPattern`—are always drawn

by the blitter using the screen density to improve the appearance of gray fills. Custom patterns, however, are stretched as appropriate by the blitter based on the destination density.

Compatibility The `lightGrayPattern` and `darkGrayPattern` values are defined only if the High-Density Display Feature Set is present.

UnderlineModeType Enum

Purpose The `UnderlineModeType` enumerated type specifies possible values for the underline mode stored in `DrawStateType`.

Prototype

```
typedef enum { noUnderline, grayUnderline,
solidUnderline, colorUnderline }
UnderlineModeType;
```

Constants

`noUnderline`
: No underline.

`grayUnderline`
: Underline is drawn using a dotted line in the current foreground color.

`solidUnderline`
: Underline is drawn using a solid line in the foreground color.

`colorUnderline`
: Underline is drawn using a solid line in the foreground color.

Compatibility The `solidUnderline` and `colorUnderline` options are only available in Palm OS 3.1 and higher.

WindowFlagsType Struct

Purpose The `WindowFlagsType` specifies different window attributes.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `WindowFlagsType` bit field. Access it only through the functions described below. Never access its bit field members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct WindowFlagsType {
  UInt16 format:1;
  UInt16 offscreen:1;
  UInt16 modal:1;
  UInt16 focusable:1;
  UInt16 enabled:1;
  UInt16 visible:1;
  UInt16 dialog:1;
  UInt16 freeBitmap:1;
  UInt16 reserved :8;
} WindowFlagsType;
```

Fields

`format`
: If set, use the `genericFormat`. If 0, use `screenFormat`.

  Screen format is the native format of the video system; windows in this format can be copied to the display faster. The generic format is device-independent. A window cannot be enabled (that is, accept pen input) unless it uses screen format.

`offscreen`
: If set, the window is offscreen. If 0, the window is onscreen.

`modal`
: If set, the window is modal. If 0, the window is not modal. You set this value when you create the window. This value is returned by `WinModal()`.

`focusable`
: If set, the window can accept the focus. If 0, the window does not accept the focus. You set this value when you create the window.

`enabled`
: If set, the window is enabled. If 0, the window is disabled.

`visible`
: If set, the window is visible if it is onscreen. If 0, the window is not visible.

`dialog`
: If set, the window is a form. If 0, the window is not a form. The `FrmInitForm()` function sets this value.

`freeBitmap`
: If set, free the bitmap when the window is freed. If 0, retain the bitmap after the window is freed.

`reserved`
: Reserved for future use. Must be 0.

Compatibility

In OS versions previous to 3.5, the `freeBitmap` flag was not present. Instead, a `compressed` flag was present, where 0 specified uncompressed and 1 specified compressed. This compressed flag is now part of the `BitmapType`.

WindowFormatType Enum

Purpose

Use this enumeration to specify the window format when creating an offscreen window with the `WinCreateOffscreenWindow()` function.

Prototype

```
typedef enum {
  screenFormat = 0,
  genericFormat,
  nativeFormat
} WindowFormatType;
```

Constants

`screenFormat`
: The window's bitmap is allocated using the hardware screen's depth, but for backward compatibility the bitmap associated with the offscreen window is always low density, and the window always uses a coordinate system that directly maps offscreen pixels to coordinates.

`genericFormat`
: Like `screenFormat`, except that `genericFormat` offscreen windows do not accept pen input.

`nativeFormat`
: Reflects the actual hardware screen format in all ways, including screen depth, density, and pixel format. Applications must always use the graphic API when drawing to a `nativeFormat` offscreen window: directly accessing offscreen pixels will produce undefined results. When using this format, the width and height arguments must be specified using the active coordinate system. Like

`genericFormat`, `nativeFormat` offscreen windows do not accept pen input.

Compatibility `WindowFormatType` is defined only if the High-Density Display Feature Set is present.

WindowType Struct

Purpose The `WindowType` structure represents a window.

WARNING! PalmSource, Inc. does not support or provide backward compatibility for the `WindowType` structure. Access it only through the functions described below. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

Prototype

```
typedef struct WindowType {
  Coord displayWidthV20;
  Coord displayHeightV20;
  void *displayAddrV20;
  WindowFlagsType windowFlags;
  RectangleType windowBounds;
  AbsRectType clippingBounds;
  BitmapPtr bitmapP;
  FrameBitsType frameType;
  DrawStateType *drawStateP;
  struct WindowType *nextWindow;
} WindowType;
```

Fields

`displayWidthV20`
: Width of the window in pre OS 3.5 devices. In OS 3.5, use `WinGetBounds()` to return the window width.

`displayHeightV20`
: Height of the window in pre OS 3.5 devices. In OS 3.5, use `WinGetBounds()` to return the window height.

`displayAddrV20`
: Pointer to the window display memory buffer in pre OS 3.5 devices. In OS 3.5 or later, call `WinGetBitmap()` and then `BmpGetBits()` to obtain the window's memory buffer.

  Writing directly to screen memory will not be supported in all devices.

`windowFlags`
: Window attributes (see `WindowFlagsType`).

`windowBounds`
: Display-relative bounds of the window. Use `WinGetBounds()` and `WinSetBounds()` to retrieve and set this value.

`clippingBounds`
: Bounds for clipping any drawing within the window. Use `WinGetClip()` and `WinSetClip()` to retrieve and set this value.

`bitmapP`
: Pointer to the window bitmap, which holds the window's contents. Use `WinGetBitmap()` to retrieve this value.

`frameType`
: Frame attributes; see `FrameBitsType`.

`drawStateP`
: Pointer to a state of the current transfer mode, pattern mode, font, underline mode, and colors. See `DrawStateType`.

  Only one drawing state exists in the system. Each window points to the same structure.

`nextWindow`
: Pointer to the next window in a linked list of windows. This linked list of windows is called the active window list. Do not change this field.

Compatibility

In OS versions previous to 3.5, this structure is slightly different. Specifically, the `bitmapP` field is instead a `viewOrigin` field of type `PointType` and specified the window origin point on the display. The `drawStateP` was named `gstate` and was of type `GraphicStatePtr`. The complete definition is shown below:

```
typedef struct WinTypeStruct {
  Word                    displayWidth;
```

```
  Word                    displayHeight;
  VoidPtr                 displayAddr;
  WindowFlagsType         windowFlags;
  RectangleType           windowBounds;
  AbsRectType             clippingBounds;
  PointType               viewOrigin;
  FrameBitsType           frameType;
  GraphicStatePtr         gstate;
  struct WinTypeStruct      *nextWindow;
} WindowType;
```

WinDrawOperation Enum

Purpose The `WinDrawOperation` enumerated type specifies the transfer mode for color drawing. This type is used as a parameter to the `WinCopyRectangle()` and `WinSetDrawMode()` functions.

Prototype

```
typedef enum {winPaint, winErase, winMask,
winInvert, winOverlay, winPaintInverse,
winSwap} WinDrawOperation;
```

Constants

`winPaint`
: Write color-matched source pixels to the destination. If a bitmap's `hasTransparency` flag is set, `winPaint` behaves like `winOverlay` instead.

`winErase`
: Write `backColor` if the source pixel is transparent.

`winMask`
: Write `backColor` if the source pixel is not transparent.

`winInvert`
: Bitwise XOR the color-matched source pixel onto the destination. This mode does not honor the transparent color in any way.

`winOverlay`
: Write color-matched source pixel to the destination if the source pixel is not transparent. Transparent pixels are skipped. For a 1-bit display, the "off" bits are considered to be the transparent color. Note that this definition of `winOverlay` is new for Palm OS Garnet; in Palm OS 4.x, the destination is set (only) where the source pixels are "on."

`winPaintInverse`
: Invert the source pixel color and then proceed as with `winPaint`.

`winSwap`
: Swap the `backColor` and `foreColor` destination colors if the source is a pattern (the type of pattern is disregarded). If the source is a bitmap, then the bitmap is transferred using `winPaint` mode instead.

Note that 2-bit, 4-bit, and 8-bit source bitmaps that don't have a color table inherit the system default color table for their given depth. 1-bit sources (bitmap, text, and patterns) that don't have a color table are given a color table where entry 0 is the `backColor` and entry 1 is the `foreColor` (`textColor` for text).

`winSwap` is not a color invert operation, although a pair of `winSwap` operations will restore the original graphics data. This mode is used by the OS to select and deselect areas of the screen. It changes destination pixels matching the foreground color to the background color, and changes destination pixels matching the background color to the foreground color. It is a mode available for rectangles, lines, and pixels, but not text or bitmaps. This mode ignores the current pattern.

The Transparent Color

As of Palm OS 4.0, bitmaps have a `hasTransparency` flag and may designate a transparent color. These concepts are augmented somewhat in Palm OS Garnet to make the transfer modes more consistent:

- Bitmaps that don't specify any transparent color (text, patterns, and version 0 bitmaps) are assumed to have a transparent color of index 0 and the `hasTransparency` bit is assumed to be `false`.
- When the `hasTransparency` flag is set and the transfer mode is `winPaint`, only the non-transparent pixels are copied to the destination. With text and patterns, the Palm OS assumes that the "off" bits are the ones designated as transparent and acts as if the `hasTransparency` flag is always false. This assumption retains backwards compatibility and unifies the use of transparency across all source data.

Compatibility This type is implemented only if 3.5 New Feature Set is present. In earlier releases, this type is named `ScrOperation` and its values begin with the prefix `scr` rather than `win`. `WinDrawOperation` is fully compatible with `ScrOperation`. Transparency is only available if 4.0 New Feature Set is present, and as mentioned in "The Transparent Color" the behavior of the window drawing modes—`winOverlay` in particular—changes slightly if 5.0 New Feature Set is present.

WinHandle Typedef

Purpose The `WinHandle` type is a pointer to a `WindowType` structure. Note that this may change.

Prototype

```
typedef WindowType *WinHandle;
```

WinLineType Struct

Purpose The `WinLineType` structure defines a line.

Prototype

```
typedef struct WinLineType {
  Coord x1;
  Coord y1;
  Coord x2;
  Coord y2;
} WinLineType;
```

Fields

`x1`
: X coordinate of the first endpoint of the line.

`y1`
: Y coordinate of the first endpoint of the line.

`x2`
: X coordinate of the second endpoint of the line.

`y2`
: Y coordinate of the second endpoint of the line.

Compatibility This type is implemented only if 3.5 New Feature Set is present.

WinPtr Typedef

Purpose The `WinPtr` type is a pointer to a `WindowType` structure.

Prototype

```
typedef WindowType *WinPtr;
```

Window Constants

If the 5.0 New Feature Set or the High-Density Display Feature Set are present, the following constants are defined:

- Window Coordinate System Constants

Window Coordinate System Constants

These constants specify the coordinate system to be used when drawing within a given window:

Constant	Value	Description
`kCoordinatesNative`	`0`	Use the bitmap's native coordinate system; this enables a 1-to-1 correspondence between coordinates and pixels.
`kCoordinatesStandard`	`72`	The coordinate system used by most handhelds running Palm OS 4.0 and earlier. On a single-density handheld, there is one screen pixel per standard coordinate. On a high-density screen, there is more than one screen pixel per standard coordinate.
`kCoordinatesOneAndAHalf`	`108`	One and a half times the standard coordinate system.
`kCoordinatesDouble`	`144`	Twice the standard coordinate system.
`kCoordinatesTriple`	`216`	Three times the standard coordinate system.
`kCoordinatesQuadruple`	`288`	Four times the standard coordinate system.

IMPORTANT: Not all coordinate systems listed in the above table are supported in this version of the High-Density Display Feature Set. For Palm OS Garnet, only `kCoordinatesNative`, `kCoordinatesStandard`, and `kCoordinatesDouble` are supported.

Pass one of these constants as an argument to `WinSetCoordinateSystem()`. These values are returned by `WinGetCoordinateSystem()`.

Compatibility Defined only if the High-Density Display Feature Set is present.

Window Functions

WinClipRectangle Function

Purpose Shrink the rectangle to make it fit within the clipping region of the current draw window.

Declared In `Window.h`

Prototype

```
void WinClipRectangle (RectangleType *rP)
```

Parameters

↔ *rP*
Pointer to a structure holding the rectangle to clip. The rectangle returned is the intersection of the rectangle passed and the clipping bounds of the draw window.

Returns Returns nothing.

Comments This function does not change the clipping rectangle of the window. To modify the window's clipping rectangle, use the `WinSetClip()` and `WinResetClip()` functions.

The draw window is the window to which all drawing functions send their output. It is returned by `WinGetDrawWindow()`.

See Also `WinCopyRectangle()`, `WinDrawRectangle()`, `WinEraseRectangle()`, `WinGetClip()`

WinCopyRectangle Function

Purpose Copy a rectangular region from one place to another (either between windows or within a single window).

Declared In `Window.h`

Prototype

```
void WinCopyRectangle (WinHandle srcWin,
   WinHandle dstWin,
   const RectangleType *srcRect,
   Coord destX, Coord destY,
   WinDrawOperation mode)
```

Parameters

→ *srcWin*
Window from which the rectangle is copied. If `NULL`, use the draw window.

→ *dstWin*
Window to which the rectangle is copied. If `NULL`, use the draw window.

→ *srcRect*
Bounds of the region to copy.

→ *destX*
Left bound of the rectangle in destination window.

→ *destY*
Top bound of the rectangle in destination window.

→ *mode*
The method of transfer from the source to the destination window (see `WinDrawOperation`).

Returns Returns nothing.

Comments Copies the bits of the window inside the rectangle region.

If the destination bitmap is compressed, the mode parameter must be `winPaint`, and the destination coordinates must be (0,0). If the width of the destination rectangle is less than 16 pixels or if the destination coordinates are not (0,0), then this function turns off compression for the destination bitmap. Normally, you do not copy to a compressed bitmap. Instead, you copy to an uncompressed bitmap and compress it afterwards.

Compatibility In OS versions before 3.5, the `mode` parameter was defined as type `ScrOperation`. It is defined as type `WinDrawOperation` only if

3.5 New Feature Set is present. `ScrOperation` and `WinDrawOperation` are fully compatible with each other.

In OS versions before 3.5, it was common practice to render a bitmap in an offscreen window and then use `WinCopyRectangle` to draw it on the screen. In version 3.5 and higher, the preferred method of doing this is to use `WinDrawBitmap()` or `WinPaintBitmap()`.

See Also `WinDrawBitmap()`

WinCreateBitmapWindow Function

Purpose Create a new offscreen window.

Declared In `Window.h`

Prototype

```
WinHandle WinCreateBitmapWindow
    (BitmapType *bitmapP, UInt16 *error)
```

Parameters

→ *bitmapP*
: Pointer to a bitmap to associate with the window. (See `BitmapType`.)

← *error*
: Pointer to any error this function encounters.

Returns Returns the handle of the new window upon success, or `NULL` if an error occurs. The `error` parameter contains one of the following:

`errNone`
: No error.

`sysErrParamErr`
: The `bitmapP` parameter is invalid. The bitmap must be uncompressed and it must have a valid pixel size (1, 2, 4, or 8). It must not be the screen bitmap.

`sysErrNoFreeResource`
: There is not enough memory to allocate a new window structure.

Comments Use `WinCreateBitmapWindow` if you want to draw into a previously created bitmap, such as a bitmap created using `BmpCreate()`.

This function generates a window wrapper for the specified bitmap. The newly created window is offscreen, uses the generic format (for device independence), and is added to the active window list. Use `WinSetDrawWindow()` to make it the draw window, and then use the window drawing functions to modify the bitmap.

When you use this function to create a window and then delete the window with `WinDeleteWindow()`, the bitmap is *not* freed when the window is freed.

`WinCreateOffscreenWindow()` uses this function to create its offscreen window. If you call `WinCreateOffscreenWindow` instead of using this function, the bitmap is freed when `WinDeleteWindow` is called.

The bitmap data will not be blitted properly if the depth of the screen is changed using `WinScreenMode()` and the new window uses a bitmap that does not define the bitmap's color table. See `WinScreenMode()` for information on how to work around this limitation.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinCreateWindow()`, `WinCreateOffscreenWindow()`

WinCreateOffscreenWindow Function

Purpose Create a new offscreen window and add it to the window list.

Declared In `Window.h`

Prototype

```
WinHandle WinCreateOffscreenWindow (Coord width,
    Coord height, WindowFormatType format,
    UInt16 *error)
```

Parameters

→ *width*
Width of the window in pixels. The coordinate system you use for this parameter depends upon the value of `format`.

→ *height*
Height of the window in pixels. The coordinate system you use for this parameter depends upon the value of `format`.

→ *format*
One of the window formats defined by `WindowFormatType`.

← *error*
Pointer to any error this function encounters.

Returns Returns the handle of the new window upon success, or `NULL` if an error occurs. The `error` parameter contains one of the following:

`errNone`
No error.

`sysErrParamErr`
The `width` or `height` parameter is `NULL` or the current color table is invalid.

`sysErrNoFreeResource`
There is not enough memory to complete the function.

The debug ROM gives a warning if you try to draw to a bad window address.

Comments Windows created with this routine draw to a memory buffer instead of the display. Use this function for temporary drawing operations such as double-buffering or save-behind operations.

The memory buffer has two formats: screen format and generic format. Screen format is the native format of the video system; windows in this format can be copied to the display faster. The generic format is device-independent. A window cannot be enabled (that is, accept pen input) unless it uses screen format.

This function differs from `WinCreateBitmapWindow()` in the following ways:

- `WinCreateOffscreenWindow` creates a new bitmap in the same depth as the current screen. `WinCreateBitmapWindow` uses the bitmap you pass in, which may or may not be in the same depth as the current screen.
- `WinCreateOffscreenWindow` uses the screen format you specify. `WinCreateBitmapWindow` always uses `genericFormat` for the format argument.
- When you delete the window created with `WinCreateOffscreenWindow`, its bitmap is freed along with the window. The bitmap used in the `WinCreateBitmapWindow` is not freed when the window is freed.

Note that if you aren't directly accessing the bits of an offscreen window's bitmap but are just using the APIs, you can always pass `nativeFormat` for the screen format even on pre-Palm OS Garnet handhelds and things will work as expected. If you need direct access to the bits of the offscreen window's bitmap, however, call `BmpCreate()` and then call `WinCreateBitmapWindow`. Because you created the bitmap, you know its format and thus can safely manipulate its bits. Calling `WinCreateOffscreenWindow` with a `format` argument of `nativeFormat` can result in a bitmap with an unexpected format: the endianness, number of bits per pixel, and so on would match the screen and therefore be fastest to draw, but your application wouldn't be able to manipulate the pixels directly.

The bitmap data will not be blitted properly if the depth of the screen is changed using `WinScreenMode()` and the new window uses a bitmap that does not define the bitmap's color table. See `WinScreenMode()` for information on how to work around this limitation.

See Also `WinCreateWindow()`, `WinScreenMode()`,

WinCreateWindow Function

Purpose Create a new window and add it to the window list.

Declared In `Window.h`

Prototype

```
WinHandle WinCreateWindow
    (const RectangleType *bounds, FrameType frame,
    Boolean modal, Boolean focusable,
    UInt16 *error)
```

Parameters

→ *bounds*
Display-relative bounds of the window.

→ *frame*
Type of frame around the window (see `FrameType`).

→ *modal*
`true` if the window is modal.

→ *focusable*
`true` if the window can be the active window.

← *error*
Pointer to any error encountered by this function.

Returns Returns the handle of the new window upon success, or `NULL` if an error occurs. The `error` parameter contains one of the following:

`errNone`
No error.

`sysErrNoFreeResource`
There is not enough memory to complete the operation.

Comments Windows created by this routine draw to the display. See `WinCreateOffscreenWindow()` for information on drawing off screen.

You typically don't call this function directly. Instead, you use `FrmInitForm()` to create form windows from a resource. Forms are much more flexible and have better system support. All forms are windows, but not all windows are forms.

The window is created with the bounds and frame type that you specify and uses the bitmap and drawing state of the current draw window. Its clipping region is reset according to the bounds you specify.

All window flags are set to 0 except for the modal and focusable flags, which you pass as a parameter to this function. Specifically, newly created windows are disabled and invisible. You must specifically enable the window before the window can accept input. You can do so with `WinSetActiveWindow()`.

See Also `WinDeleteWindow()`

WinDeleteWindow Function

Purpose Remove a window from the window list and free the memory used by the window.

Declared In `Window.h`

Prototype

```
void WinDeleteWindow (WinHandle winHandle,
   Boolean eraseIt)
```

Parameters → *winHandle*
Handle of window to delete.

→ *eraseIt*
If `true`, the window is erased before it is deleted. If `false`, the window is not erased.

Returns Returns nothing.

Comments This function frees all memory associated with the window. Windows created using `WinCreateOffscreenWindow()` have their bitmaps freed; windows created using `WinCreateWindow()` or `WinCreateBitmapWindow()` do not.

The `eraseIt` parameter affects onscreen windows only; offscreen windows are never erased. As a performance optimization, you might set `eraseIt` to `false` for an onscreen window if you know that you are going to immediately redraw the area anyway. For example, when the form manager closes a form dialog, it restores the area with the save-behind bits it had stored for that form. For this reason, when the form manager deletes the dialog window, it passes `false` for `eraseIt` because the entire area will be redrawn.

WinDisplayToWindowPt Function

Purpose Convert a display-relative coordinate to a window-relative coordinate. The coordinate returned is relative to the display window.

Declared In `Window.h`

Prototype

```
void WinDisplayToWindowPt (Coord *extentX,
    Coord *extentY)
```

Parameters

↔ *extentX*
Pointer to x coordinate to convert.

↔ *extentY*
Pointer to y coordinate to convert.

Returns Returns nothing.

See Also WinWindowToDisplayPt()

WinDrawBitmap Function

Purpose Draw a bitmap at the given coordinates in `winPaint` mode (see `WinDrawOperation` for mode details).

Declared In `Window.h`

Prototype

```
void WinDrawBitmap (BitmapPtr bitmapP, Coord x,
    Coord y)
```

Parameters

→ *bitmapP*
Pointer to a bitmap.

→ *x*
The x coordinate of the top-left corner.

→ *y*
The y coordinate of the top-left corner.

Returns Returns nothing.

Comments If the bitmap has multiple depths (is a bitmap family), the closest match less than or equal to the current draw window depth is used. If such a bitmap does not exist, the bitmap with the closest match greater than the draw window depth is used.

If the bitmap has its own color table, color conversion to the draw window color table will be applied (on OS 3.5 or later). This color conversion is slow and not recommended. Instead of including a color table in the bitmap, consider using `WinPalette()` to change the system color table, draw the bitmap, and then change the system color table back when the bitmap is no longer visible.

This function differs from `WinPaintBitmap()` in that this function always uses `winPaint` mode (copy mode) as the transfer mode. `WinPaintBitmap` uses the current drawing state transfer mode.

See Also `WinEraseRectangle()`

WinDrawChar Function

Purpose Draw the specified character in the draw window.

Declared In `Window.h`

Prototype

```
void WinDrawChar (WChar theChar, Coord x,
    Coord y)
```

Parameters

→ *theChar*
The character to draw. This may be either a single-byte character or a multi-byte character.

→ *x*
x coordinate of the location where the character is to be drawn (left bound).

→ *y*
y coordinate of the location where the character is to be drawn (top bound).

Returns Returns nothing.

Comments Before calling this function, call `WinSetUnderlineMode()` and `FntSetFont()` to set the desired underline and font to draw the characters.

This function differs from `WinPaintChar()` in that this function always uses `winPaint` mode (see `WinDrawOperation`). This means the on bits are drawn in the text color, the off bits are in the background color, and underlines are in the foreground color. `WinPaintChar` uses the current drawing state transfer mode instead of `winPaint`.

Compatibility Implemented only if 3.1 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `WinGlueDrawChar`. For more information, see Chapter 80, "PalmOSGlue Library."

NOTE: Though this function is new in the 3.1 New Feature Set, due to a bug in the 3.1 implementation, when running on Palm OS 3.1 you must link with the PalmOSGlue library and call `WinGlueDrawChar`, rather than calling `WinDrawChar`.

See Also `WinDrawChars()`, `WinDrawInvertedChars()`, `WinDrawTruncChars()`, `WinEraseChars()`, `WinInvertChars()`, `WinPaintChars()`

WinDrawChars Function

Purpose Draw the specified characters in the draw window.

Declared In `Window.h`

Prototype

```
void WinDrawChars (const Char *chars, Int16 len,
    Coord x, Coord y)
```

Parameters

→ *chars*
Pointer to the characters to draw.

→ *len*
Length in bytes of the characters to draw.

→ *x*
x coordinate of the first character to draw (left bound).

→ *y*
y coordinate of the first character to draw (top bound).

Returns Returns nothing.

Comments This function is useful for printing non-editable status or warning messages on the screen.

Before calling this function, call `WinSetUnderlineMode()` and `FntSetFont()` to set the desired underline and font to draw the characters.

This function differs from `WinPaintChars()` in that this function always uses `winPaint` mode (see `WinDrawOperation`). This means the on bits are drawn in the text color, the off bits are in the background color, and underlines are in the foreground color.

`WinPaintChar` uses the current drawing state transfer mode instead of `winPaint`.

See Also `WinDrawChar()`, `WinDrawInvertedChars()`, `WinDrawTruncChars()`, `WinEraseChars()`, `WinInvertChars()`, `WinPaintChar()`

WinDrawGrayLine Function

Purpose Draw a dashed line in the draw window.

Declared In `Window.h`

Prototype

```
void WinDrawGrayLine (Coord x1, Coord y1,
    Coord x2, Coord y2)
```

Parameters

→ *x1*
x coordinate of line start point.

→ *y1*
y coordinate of line start point.

→ *x2*
x coordinate of line endpoint.

→ *y2*
y coordinate of line endpoint.

Returns Returns nothing.

Comments This routine does not draw in the gray color; it draws with alternating foreground and background pixels. That is, it uses the `grayPattern` pattern type.

See Also `WinDrawLine()`, `WinEraseLine()`, `WinFillLine()`, `WinInvertLine()`, `WinPaintLine()`, `WinPaintLines()`

WinDrawGrayRectangleFrame Function

Purpose Draw a gray rectangular frame in the draw window.

Declared In `Window.h`

Prototype

```
void WinDrawGrayRectangleFrame (FrameType frame,
    const RectangleType *rP)
```

Parameters

→ *frame*
Type of frame to draw (see `FrameType`).

→ *rP*
Pointer to the rectangle to frame.

Returns Returns nothing.

Comments This routine does not draw in the gray color; it draws with alternating foreground and background pixels. The standard gray pattern is not used by this routine; rather, the frame is drawn so that the top-left pixel of the frame is always on.

See Also `WinDrawRectangleFrame()`, `WinEraseRectangleFrame()`, `WinGetFramesRectangle()`, `WinInvertRectangleFrame()`, `WinPaintRectangleFrame()`

WinDrawInvertedChars Function

Purpose Draw the specified characters inverted (background color) in the draw window.

Declared In `Window.h`

Prototype

```
void WinDrawInvertedChars (const Char *chars,
    Int16 len, Coord x, Coord y)
```

Parameters

→ *chars*
Pointer to the characters to draw.

→ *len*
Length in bytes of the characters to draw.

→ *x*
x coordinate of the first character to draw (left bound).

→ *y*
y coordinate of the first character to draw (top bound).

Returns Returns nothing.

Comments This routine draws the *on* bits and any underline in the background color and the *off* bits in the text color. (Black and white uses copy NOT mode.) This is the standard function for drawing inverted text.

Before calling this function, consider calling `WinSetUnderlineMode()` and `FntSetFont()`.

See Also `WinDrawChar()`, `WinDrawChars()`, `WinDrawTruncChars()`, `WinEraseChars()`, `WinInvertChars()`, `WinPaintChar()`, `WinPaintChars()`

WinDrawLine Function

Purpose Draw a line in the draw window using the current foreground color.

Declared In `Window.h`

Prototype

```
void WinDrawLine (Coord x1, Coord y1, Coord x2,
    Coord y2)
```

Parameters

→ *x1*
x coordinate of line start point.

→ *y1*
y coordinate of line start point.

→ *x2*
x coordinate of line endpoint.

→ *y2*
y coordinate of line endpoint.

Returns Returns nothing.

Comments This function differs from `WinPaintLine()` in that it always uses `winPaint` mode (see `WinDrawOperation`). `WinPaintLine` uses the current drawing state transfer mode instead of `winPaint`.

See Also `WinDrawGrayLine()`, `WinEraseLine()`, `WinFillLine()`, `WinInvertLine()`, `WinPaintLine()`, `WinPaintLines()`

WinDrawPixel Function

Purpose Draw a pixel in the draw window using the current foreground color.

Declared In `Window.h`

Prototype

```
void WinDrawPixel (Coord x, Coord y)
```

Parameters

→ *x*
Pointer to the x coordinate of a pixel.

→ *y*
Pointer to the y coordinate of a pixel.

Returns Returns nothing. May display a fatal error message if the draw window's bitmap is compressed.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinErasePixel()`, `WinInvertPixel()`, `WinPaintPixel()`, `WinPaintPixels()`

WinDrawRectangle Function

Purpose Draw a rectangle in the draw window using the current foreground color.

Declared In `Window.h`

Prototype

```
void WinDrawRectangle (const RectangleType *rP,
    UInt16 cornerDiam)
```

Parameters

→ *rP*
Pointer to the rectangle to draw.

→ *cornerDiam*
Radius of rounded corners. Specify zero for square corners.

Returns Returns nothing.

Comments The `cornerDiam` parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides.

This function differs from `WinPaintRectangle()` in that it always uses `winPaint` mode (see `WinDrawOperation`).

`WinPaintRectangle` uses the current drawing state transfer mode instead of `winPaint`.

See Also `WinEraseRectangle()`, `WinFillRectangle()`, `WinInvertRectangle()`

WinDrawRectangleFrame Function

Purpose Draw a rectangular frame in the draw window using the current foreground color.

Declared In `Window.h`

Prototype

```
void WinDrawRectangleFrame (FrameType frame,
    const RectangleType *rP)
```

Parameters

→ *frame*
Type of frame to draw (see `FrameType`).

→ *rP*
Pointer to the rectangle to frame.

Returns Returns nothing.

Comments The frame is drawn outside the specified rectangle.

This function differs from `WinPaintRectangleFrame()` in that it always uses `winPaint` mode (see `WinDrawOperation`). `WinPaintRectangleFrame` uses the current drawing state transfer mode instead of `winPaint`.

See Also `WinDrawGrayRectangleFrame()`, `WinEraseRectangleFrame()`, `WinGetFramesRectangle()`, `WinInvertRectangleFrame()`

WinDrawTruncChars Function

Purpose Draw the specified characters in the draw window, truncating the characters to the specified width.

Declared In `Window.h`

Prototype

```
void WinDrawTruncChars (const Char *chars,
    Int16 len, Coord x, Coord y, Coord maxWidth)
```

Parameters

→ *chars*
Pointer to the characters to draw.

→ *len*
Length in bytes of the characters to draw.

→ *x*
x coordinate of the first character to draw (left bound).

→ *y*
y coordinate of the first character to draw (top bound).

→ *maxWidth*
Maximum width in pixels of the characters that are to be drawn.

Returns Returns nothing.

Comments Before calling this function, consider calling `WinSetUnderlineMode()` and `FntSetFont()`.

If drawing all of the specified characters requires more space than `maxWidth` allows, `WinDrawTruncChars` draws one less than the number of characters that can fit in `maxWidth` and then draws an ellipsis (...) in the remaining space. (If the boundary characters are narrower than the ellipsis, more than one character may be dropped to make room.) If `maxWidth` is narrower than the width of an ellipsis, nothing is drawn.

Use this function to truncate text that may contain multi-byte characters.

Compatibility Implemented only if 3.1 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `WinGlueDrawTruncChars`. For more information, see Chapter 80, "PalmOSGlue Library."

NOTE: Though this function is new in the 3.1 New Feature Set, due to a bug in the 3.1 implementation, when running on Palm OS 3.1 you must link with the PalmOSGlue library and call `WinGlueDrawTruncChars`, rather than calling `WinDrawTruncChars`.

See Also `WinDrawChar()`, `WinDrawChars()`, `WinDrawInvertedChars()`, `WinEraseChars()`, `WinInvertChars()`, `WinPaintChar()`, `WinPaintChars()`

WinEraseChars Function

Purpose Erase the specified characters in the draw window.

Declared In `Window.h`

Prototype

```
void WinEraseChars (const Char *chars, Int16 len,
    Coord x, Coord y)
```

Parameters

→ *chars*
Pointer to the characters to erase.

→ *len*
Length in bytes of the characters to erase.

→ *x*
x coordinate of the first character to erase (left bound).

→ *y*
y coordinate of the first character to erase (top bound).

Returns Returns nothing.

Comments The `winMask` transfer mode is used to erase the characters. See `WinDrawOperation` for more information. This has the effect of erasing only the on bits for the characters rather than the entire text rectangle. This function only works if the foreground color is black and the background color is white.

See Also WinDrawChar(), WinDrawChars(), WinDrawInvertedChars(), WinDrawTruncChars(), WinInvertChars(), WinPaintChar(), WinPaintChars()

WinEraseLine Function

Purpose Draw a line in the draw window using the current background color.

Declared In `Window.h`

Prototype

```
void WinEraseLine (Coord x1, Coord y1, Coord x2,
    Coord y2)
```

Parameters

→ *x1*
x coordinate of line start point.

→ *y1*
y coordinate of line start point.

→ *x2*
x coordinate of line endpoint.

→ *y2*
y coordinate of line endpoint.

Returns Returns nothing.

See Also WinDrawGrayLine(), WinDrawLine(), WinFillLine(), WinInvertLine(), WinPaintLine(), WinPaintLines()

WinErasePixel Function

Purpose Draw a pixel in the draw window using the current background color.

Declared In `Window.h`

Prototype

```
void WinErasePixel (Coord x, Coord y)
```

Parameters

→ *x*
Pointer to the x coordinate of a pixel.

→ *y*
Pointer to the y coordinate of a pixel.

Returns Returns nothing.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinDrawPixel(), WinInvertPixel(), WinPaintPixel(), WinPaintPixels()

WinEraseRectangle Function

Purpose Draw a rectangle in the draw window using the current background color.

Declared In `Window.h`

Prototype

```
void WinEraseRectangle (const RectangleType *rP,
    UInt16 cornerDiam)
```

Parameters

→ *rP*
Pointer to the rectangle to erase.

→ *cornerDiam*
Radius of rounded corners. Specify zero for square corners.

Returns Returns nothing.

Comments The `cornerDiam` parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides.

See Also WinDrawRectangle(), WinFillRectangle(), WinInvertRectangle(), WinPaintRectangle()

WinEraseRectangleFrame Function

Purpose Draw a rectangular frame in the draw window using the current background color.

Declared In `Window.h`

Prototype

```
void WinEraseRectangleFrame (FrameType frame,
    const RectangleType *rP)
```

Parameters

→ *frame*
Type of frame to draw (see `FrameType`).

→ *rP*
Pointer to the rectangle to frame.

Returns Returns nothing.

See Also WinDrawGrayRectangleFrame(), WinDrawRectangleFrame(), WinGetFramesRectangle(), WinInvertRectangleFrame(), WinPaintRectangleFrame()

WinEraseWindow Function

Purpose Erase the contents of the draw window.

Declared In `Window.h`

Prototype

```
void WinEraseWindow (void)
```

Parameters `None.`

Returns Returns nothing.

Comments `WinEraseRectangle()` is used to erase the window. This routine doesn't erase the frame around the draw window. See `WinEraseRectangleFrame()` and `WinGetWindowFrameRect()`.

WinFillLine Function

Purpose Fill a line in the draw window with the current pattern.

Declared In `Window.h`

Prototype

```
void WinFillLine (Coord x1, Coord y1, Coord x2,
    Coord y2)
```

Parameters

→ *x1*
: x coordinate of line start point.

→ *y1*
: y coordinate of line start point.

→ *x2*
: x coordinate of line endpoint.

→ *y2*
: y coordinate of line endpoint.

Returns Returns nothing.

Comments You can set the current pattern with `WinSetPattern()`.

See Also WinDrawGrayLine(), WinDrawLine(), WinEraseLine(), WinInvertLine(), WinPaintLine(), WinPaintLines()

WinFillRectangle Function

Purpose Draw a rectangle in the draw window with current pattern.

Declared In `Window.h`

Prototype

```
void WinFillRectangle (const RectangleType *rP,
   UInt16 cornerDiam)
```

Parameters

→ *rP*
Pointer to the rectangle to draw.

→ *cornerDiam*
Radius of rounded corners. Specify zero for square corners.

Returns Returns nothing.

Comments You can set the current pattern with `WinSetPattern()`.

The `cornerDiam` parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides.

See Also WinDrawRectangle(), WinEraseRectangle(), WinInvertRectangle(), WinPaintRectangle()

WinGetActiveWindow Function

Purpose Return the window handle of the active window.

Declared In `Window.h`

Prototype

```
WinHandle WinGetActiveWindow (void)
```

Parameters `None.`

Returns Returns the handle of the active window. All user input is directed to the active window.

See Also `WinSetActiveWindow()`, `WinGetDisplayWindow()`, `WinGetFirstWindow()`, `WinGetDrawWindow()`

WinGetBitmap Function

Purpose Return a pointer to a window's bitmap, which holds the window contents.

Declared In `Window.h`

Prototype

```
BitmapType *WinGetBitmap (WinHandle winHandle)
```

Parameters → *winHandle*
Handle of window from which to get the bitmap.

Returns Returns a pointer to the bitmap or `NULL` if `winHandle` is invalid.

Comments For onscreen windows, the bitmap returned always represents the whole screen. Thus, the top-left corner of the returned bitmap may not be the top-left corner of the window.

Compatibility Implemented only if 3.5 New Feature Set is present.

WinGetBounds Function

Purpose Return the bounds of a window in display-relative coordinates.

Declared In `Window.h`

Prototype

```
void WinGetBounds (WinHandle winH,
   RectangleType *rP)
```

Parameters → *winH*
Handle to a window.

← *rP*
Pointer to a rectangle.

Returns Returns nothing.

Comments This function returns in `rP` the bounds of the window represented by `winH`. This corresponds to the convention used by `WinSetBounds()`, because it takes a window handle as an argument.

Compatibility The implementation of this function changed in Palm OS version 4.0. If 4.0 New Feature Set is present, it works as documented here.

Prior to Palm OS 4.0, `WinGetBounds` returned the bounds of the draw window, and did not take a window handle as an argument. If an application needed to determine the bounds of an arbitrary

window, the application would call `WinSetDrawWindow()` to temporarily set the draw window to the desired window, then `WinGetBounds` would be called to get the bounds of the draw window, then `WinSetDrawWindow()` would be called again to restore the draw window. This is no longer necessary.

As of Palm OS 4.0, an application should use `WinGetDrawWindowBounds()` to get the bounds of the draw window.

See Also WinGetWindowExtent(), WinGetDrawWindowBounds()

WinGetClip Function

Purpose Return the clipping rectangle of the draw window.

Declared In `Window.h`

Prototype

```
void WinGetClip (RectangleType *rP)
```

Parameters

← *rP*
Pointer to a structure to hold the clipping bounds.

Returns Returns nothing.

See Also WinSetClip()

WinGetCoordinateSystem Function

Purpose Get the coordinate system

Declared In `Window.h`

Prototype

```
UInt16 WinGetCoordinateSystem (void)
```

Parameters `None.`

Returns Returns a value representing the current coordinate system. See "Window Coordinate System Constants" on page 1227 for the values that this function can return.

Comments Use this function to determine the active coordinate system. Armed with this information, an application can properly initialize graphic primitive coordinates and dimensions, or can modify the coordinate system with `WinSetCoordinateSystem()`.

Compatibility Implemented only if the High-Density Display Feature Set is present.

See Also `WinSetCoordinateSystem()`

WinGetDisplayExtent Function

Purpose Return the width and height of the display (the screen).

Declared In `Window.h`

Prototype

```
void WinGetDisplayExtent (Coord *extentX,
    Coord *extentY)
```

Parameters

← *extentX*
Pointer to the width of the display in pixels.

← *extentY*
Pointer to the height of the display in pixels.

Returns Returns nothing.

WinGetDisplayWindow Function

Purpose Return the window handle of the display (screen) window.

Declared In `Window.h`

Prototype

```
WinHandle WinGetDisplayWindow (void)
```

Parameters None.

Returns Returns the handle of display window.

Comments The display window is created by the system at start-up; it has the same size as the Palm OS drawable area of the physical display (screen).

See Also WinGetDisplayExtent(), WinGetActiveWindow(), WinGetDrawWindow()

WinGetDrawWindow Function

Purpose Return the window handle of the current draw window.

Declared In `Window.h`

Prototype
```
WinHandle WinGetDrawWindow (void)
```

Parameters None.

Returns Returns handle of draw window.

See Also WinGetDisplayWindow(), WinGetActiveWindow(), WinSetDrawWindow()

WinGetDrawWindowBounds Function

Purpose Return the bounds of the draw window.

Declared In `Window.h`

Prototype
```
void WinGetDrawWindowBounds (RectangleType *rP)
```

Parameters ← *rP*
Pointer to the window bounds.

Returns Returns nothing.

Comments A pointer to the bounds of the draw window is returned. This function is equivalent to `WinGetBounds()` that was in Palm OS prior to 4.0.

See Also `WinGetBounds()`

WinGetFirstWindow Function

Purpose Return a pointer to the first window in the linked list of windows.

Declared In `Window.h`

Prototype
```
WinHandle WinGetFirstWindow (void)
```

Parameters None.

Returns Returns handle of first window.

Comments This function is usually used by the system only.

Compatibility If Palm OS Cobalt Feature Set is present this function is unimplemented.

See Also WinGetActiveWindow()

WinGetFramesRectangle Function

Purpose Return the rectangle that includes a rectangle together with the specified frame around it.

Declared In `Window.h`

Prototype

```
void WinGetFramesRectangle (FrameType frame,
    const RectangleType *rP,
    RectangleType *obscuredRect)
```

Parameters

→ *frame*
Type of rectangle frame (see `FrameType`).

→ *rP*
Pointer to the rectangle to frame.

← *obscuredRect*
Pointer to the rectangle that includes both the specified rectangle and its frame.

Returns Returns nothing.

Comments Frames are always drawn around (outside) a rectangle.

See Also WinGetWindowFrameRect(), WinGetBounds()

WinGetPattern Function

Purpose Return the current fill pattern.

Declared In `Window.h`

Prototype

```
void WinGetPattern (CustomPatternType *patternP)
```

Parameters

← *patternP*
Buffer where the current pattern is returned (see `CustomPatternType`).

Returns Returns nothing.

Comments The fill pattern is used by `WinFillLine()` and `WinFillRectangle()`.

This function returns the value of `patternData` in the current drawing state. (See `DrawStateType`.) The `patternData` field is only set if the pattern field is `customPattern`. Therefore, it's a good idea to use `WinGetPatternType()` instead of this function on systems that support `WinGetPatternType()`.

See Also WinSetPattern()

WinGetPatternType Function

Purpose Return the current pattern type.

Declared In `Window.h`

Prototype

```
PatternType WinGetPatternType (void)
```

Parameters `None.`

Returns Returns the current draw window pattern type (see `PatternType`). If the return value is `customPattern`, you can retrieve the pattern with `WinGetPattern()`.

Comments The fill pattern is used by `WinFillLine()` and `WinFillRectangle()`.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinSetPatternType()

WinGetPixel Function

Purpose Return the color value of a pixel in the current draw window.

Declared In `Window.h`

Prototype

```
IndexedColorType WinGetPixel (Coord x, Coord y)
```

Parameters

→ *x*
Pointer to the x coordinate of a pixel.

→ *y*
Pointer to the y coordinate of a pixel.

Returns Returns the indexed color value of the pixel. See `IndexedColorType`. A return value of 0 means either that the coordinates do not lie in the current draw window or that they do and the color of that pixel is index 0 (typically white).

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinIndexToRGB()`

WinGetPixelRGB Function

Purpose Return the RGB color values of a pixel in the current draw window.

Declared In `Window.h`

Prototype

```
Err WinGetPixelRGB (Coord x, Coord y,
   RGBColorType *rgbP)
```

Parameters

→ *x*
Pointer to the x coordinate of a pixel.

→ *y*
Pointer to the y coordinate of a pixel.

← *rgbP*
RGB color components of the pixel.

Returns Returns `errNone` or `sysErrParamErr`. `sysErrParamErr` is returned when the `x` or `y` arguments are < 0 or when they are outside the bounds of the draw window.

Comments The RGB color values of the pixel are returned as an `RGBColorType`. This function can be used with both indexed or direct color modes. A return value of `sysErrParamErr` means that the coordinates do not lie in the current draw window.

Compatibility Implemented only if 4.0 New Feature Set is present.

WinGetScalingMode Function

Purpose Get the current scaling mode.

Declared In `Window.h`

Prototype
```
UInt32 WinGetScalingMode (void)
```

Parameters `None.`

Returns Returns the current scaling mode. See the description of `WinSetScalingMode()` for information on interpreting the returned scaling mode value.

Compatibility Implemented only if the 1.5x Display Feature Set is present.

WinGetSupportedDensity Function

Purpose Enumerate the various display densities supported by the blitter.

Declared In `Window.h`

Prototype
```
Err WinGetSupportedDensity (UInt16 *densityP)
```

Parameters

↔ *densityP*
: Pointer to a supported density value. Set this value to zero before calling this function for the first time. Subsequent calls cause this value to be set to one of the display densities supported by the handheld.

Returns Returns `errNone` unless the value you supply in `*densityP` isn't a supported density and isn't zero, in which case this function returns `SysErrParamErr`.

Comments Initialize `*densityP` to zero before your application calls this function for the first time. Repeated calls to `WinGetSupportedDensity` will cause the value pointed to by `densityP` to change; these values represent the supported display densities, in order from low to high density. After the last supported density value, this function sets `*densityP` back to zero.

NOTE: The densities reported by this function are those that are supported by the blitter. These densities are not necessarily supported by the underlying hardware. A handheld with a low-density screen that is able to scale high-density bitmaps will report that it can handle both high and low density bitmaps. Use `WinScreenGetAttribute()` to determine the density of the handheld's screen.

Density values are defined in `Bitmap.h`; see the `DensityType` enum. Only those values supported by a given handheld will be returned by `WinGetSupportedDensity`,. For example, on a handheld with a double-density display this function returns `kDensityLow`, followed by `kDensityDouble`, followed by 0. For each supported density, the inverse scaling factor is supported. In this example, the blitter supports pixel-doubling low-density data for a double-density destination, and the blitter supports pixel-halving high-density data for a low-density destination.

The value pointed to by `densityP` should only be zero or one of the density values supported by the handheld. If it has any other value when you call `WinGetSupportedDensity`, this function will simply return `sysErrParamErr`.

Compatibility Implemented only if the High-Density Display Feature Set is present.

WinGetWindowExtent Function

Purpose Return the width and height of the current draw window.

Declared In `Window.h`

Prototype

```
void WinGetWindowExtent (Coord *extentX,
    Coord *extentY)
```

Parameters

← *extentX*
Pointer to the width in pixels of the draw window.

← *extentY*
Pointer to the height in pixels of the draw window.

Returns Returns nothing.

See Also `WinGetBounds()`, `WinGetWindowFrameRect()`,

WinGetWindowFrameRect Function

Purpose Return a rectangle, in display-relative coordinates, that defines the size and location of a window and its frame.

Declared In `Window.h`

Prototype

```
void WinGetWindowFrameRect (WinHandle winHandle,
    RectangleType *r)
```

Parameters

→ *winHandle*
Handle of window whose coordinates are desired.

← *r*
Pointer to the coordinates of the window.

Returns Returns nothing.

See Also WinGetBounds()

WinIndexToRGB Function

Purpose Convert an index in the currently active color table to an RGB value.

Declared In `Window.h`

Prototype

```
void WinIndexToRGB (IndexedColorType i,
    RGBColorType *rgbP)
```

Parameters

→ *i*
A color index value. See `IndexedColorType`.

← *rgbP*
Pointer to an RGB color value corresponding to the index value `i`. See `RGBColorType`.

Returns Returns nothing.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinRGBToIndex()

WinInvertChars Function

Purpose Invert the specified characters in the draw window.

Declared In `Window.h`

Prototype

```
void WinInvertChars (const Char *chars,
    Int16 len, Coord x, Coord y)
```

Parameters

→ *chars*
Pointer to the characters to invert.

→ *len*
Length in bytes of the characters to invert.

→ *x*
x coordinate of the first character to invert (left bound).

→ *y*
y coordinate of the first character to invert (top bound).

Returns Returns nothing.

Comments This function applies the `winInvert` operation of `WinDrawOperation` to the characters in the draw window.

To perform color inverting, use `WinSetTextColor()` and `WinSetBackColor()` to choose the desired colors, and call `WinPaintChar()`.

See Also WinDrawChar(), WinDrawChars(), WinDrawInvertedChars(), WinDrawTruncChars(), WinEraseChars(), WinPaintChar(), WinPaintChars()

WinInvertLine Function

Purpose Invert a line in the draw window (using the `WinDrawOperation` `winInvert`).

Declared In `Window.h`

Prototype

```
void WinInvertLine (Coord x1, Coord y1, Coord x2,
    Coord y2)
```

Parameters

→ *x1*
x coordinate of line start point.

→ *y1*
y coordinate of line start point.

→ *x2*
x coordinate of line endpoint.

→ *y2*
y coordinate of line endpoint.

Returns Returns nothing.

See Also WinDrawGrayLine(), WinDrawLine(), WinEraseLine(), WinFillLine(), WinPaintLine(), WinPaintLines()

WinInvertPixel Function

Purpose Invert a pixel in the draw window (using the `WinDrawOperation` `winInvert`).

Declared In `Window.h`

Prototype
```
void WinInvertPixel (Coord x, Coord y)
```

Parameters → *x*
Pointer to the x coordinate of a pixel.

→ *y*
Pointer to the y coordinate of a pixel.

Returns Returns nothing.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinDrawPixel(), WinErasePixel(), WinPaintPixel(), WinPaintPixels()

WinInvertRectangle Function

Purpose Invert a rectangle in the draw window (using the `WinDrawOperation` `winInvert`).

Declared In `Window.h`

Prototype
```
void WinInvertRectangle (const RectangleType *rP,
    UInt16 cornerDiam)
```

Parameters → *rP*
Pointer to the rectangle to invert.

→ *cornerDiam*
Radius of rounded corners. Specify zero for square corners.

Returns Returns nothing.

Comments The `cornerDiam` parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides.

The operating system itself does not use the inverting routines. Instead, it uses the `winSwap` transfer mode, or it changes the color selection and uses the `WinPaint...` routines.

See Also WinDrawRectangle(), WinEraseRectangle(), WinFillRectangle(), WinPaintRectangle()

WinInvertRectangleFrame Function

Purpose Invert a rectangular frame in the draw window (using the `WinDrawOperation` `winInvert`).

Declared In `Window.h`

Prototype

```
void WinInvertRectangleFrame (FrameType frame,
    const RectangleType *rP)
```

Parameters

→ *frame*
Type of frame to draw (see `FrameType`).

→ *rP*
Pointer to the rectangle to frame.

Returns Returns nothing.

See Also WinDrawGrayRectangleFrame(), WinDrawRectangleFrame(), WinEraseRectangleFrame(), WinGetFramesRectangle(), WinPaintRectangleFrame()

WinModal Function

Purpose Return `true` if the specified window is modal.

Declared In `Window.h`

Prototype
```
Boolean WinModal (WinHandle winHandle)
```

Parameters → *winHandle*
Handle of a window.

Returns Returns `true` if the window is modal, otherwise `false`.

Comments A window is modal if it cannot lose the focus.

See Also `FrmAlert()`, `FrmCustomAlert()`, `FrmDoDialog()`

WinPaintBitmap Function

Purpose Draw a bitmap in the current draw window at the specified coordinates with the current draw mode.

Declared In `Window.h`

Prototype
```
void WinPaintBitmap (BitmapType *bitmapP,
   Coord x, Coord y)
```

Parameters → *bitmapP*
Pointer to a bitmap.

→ *x*
The x coordinate of the top-left corner.

→ *y*
The y coordinate of the top-left corner.

Returns Returns nothing.

Comments If the bitmap has multiple depths (is a bitmap family), the closest match less than or equal to the current draw window depth is used. If such a bitmap does not exist, the bitmap with the closest match greater than the draw window depth is used.

Using `WinPaintBitmap` is now recommended instead of the previous practice of rendering bitmaps into an offscreen window and then using `WinCopyRectangle()` to draw them on screen.

The current draw mode is set by `WinSetDrawMode()`.

If the bitmap has its own color table, color conversion to the draw window color table will be applied (on OS 3.5 or later). This color conversion is slow and not recommended. Instead of including a color table in the bitmap, consider using `WinPalette()` to change the system color table, draw the bitmap, and then change the system color table back when the bitmap is no longer visible.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinDrawBitmap(), WinEraseRectangle(), WinPaintTiledBitmap()

WinPaintChar Function

Purpose Draw a character in the draw window using the current drawing state.

Declared In `Window.h`

Prototype

```
void WinPaintChar (WChar theChar, Coord x,
    Coord y)
```

Parameters

→ *theChar*
The character to draw. This may be either a single-byte character or a multi-byte character.

→ *x*
x coordinate of the location where the character is to be drawn (left bound).

→ *y*
y coordinate of the location where the character is to be drawn (top bound).

Returns Returns nothing.

See Also `WinPaintChar` draws the *on* bits in the text color and the *off* bits in the background color, with underlines (if any) drawn in the foreground color using the current drawing mode.

This function uses the current drawing state, which is stored in a `DrawStateType` structure. See the description of that structure to learn the functions you can call to set the drawing state to the values you want.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinDrawChar(), WinDrawChars(), WinDrawInvertedChars(), WinDrawTruncChars(), WinEraseChars(), WinInvertChars(), WinPaintChars()

WinPaintChars Function

Purpose Draw the specified characters in the draw window with the current draw state.

Declared In `Window.h`

Prototype

```
void WinPaintChars (const Char *chars, Int16 len,
    Coord x, Coord y)
```

Parameters

→ *chars*
Pointer to the characters to draw.

→ *len*
Length in bytes of the characters to draw.

→ *x*
x coordinate of the first character to draw (left bound).

→ *y*
y coordinate of the first character to draw (top bound).

Returns Returns nothing.

Comments `WinPaintChars` draws the *on* bits in the text color and the *off* bits in the background color, with underlines (if any) drawn in the foreground color using the current drawing mode.

This function uses the current drawing state, which is stored in a `DrawStateType` structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

Before calling this function, consider calling `WinSetUnderlineMode()` and `FntSetFont()`.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinDrawChar(), WinDrawChars(), WinDrawInvertedChars(), WinDrawTruncChars(), WinEraseChars(), WinInvertChars(), WinPaintChar()

WinPaintLine Function

Purpose Draw a line in the draw window using the current drawing state.

Declared In `Window.h`

Prototype

```
void WinPaintLine (Coord x1, Coord y1, Coord x2,
    Coord y2)
```

Parameters

→ *x1*
x coordinate of line beginning point.

→ *y1*
y coordinate of line beginning point.

→ *x2*
x coordinate of line endpoint.

→ *y2*
y coordinate of line endpoint.

Returns Returns nothing.

Comments This function uses the current drawing state, which is stored in a `DrawStateType` structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinDrawLine(), WinDrawGrayLine(), WinEraseLine(), WinFillLine(), WinInvertLine(), WinPaintLines()

WinPaintLines Function

Purpose Draw several lines in the draw window using the current drawing state.

Declared In `Window.h`

Prototype

```
void WinPaintLines (UInt16 numLines,
    WinLineType lines[])
```

Parameters

→ *numLines*
Number of lines to paint.

→ *lines*
Array of lines. See `WinLineType`.

Returns Returns nothing.

Comments This function uses the current drawing state, which is stored in a `DrawStateType` structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinDrawLine(), WinDrawGrayLine(), WinEraseLine(), WinFillLine(), WinInvertLine(), WinPaintLine()

WinPaintPixel Function

Purpose Render a pixel in the draw window using the current drawing state.

Declared In `Window.h`

Prototype

```
void WinPaintPixel (Coord x, Coord y)
```

Parameters

→ *x*
Pointer to the x coordinate of a pixel.

→ *y*
Pointer to the y coordinate of a pixel.

Returns Returns nothing.

Comments This function uses the current drawing state, which is stored in a `DrawStateType` structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinDrawPixel(), WinErasePixel(), WinInvertPixel(), WinPaintPixels()

WinPaintPixels Function

Purpose Render several pixels in the draw window using the current drawing state.

Declared In `Window.h`

Prototype

```
void WinPaintPixels (UInt16 numPoints,
    PointType pts[])
```

Parameters

→ *numPoints*
Number of pixels to paint.

→ *pts*
Array of pixels.

Returns Returns nothing.

Comments This function uses the current drawing state, which is stored in a `DrawStateType` structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinDrawPixel(), WinErasePixel(), WinInvertPixel(), WinPaintPixel()

WinPaintRectangle Function

Purpose Draw a rectangle in the draw window using the current drawing state.

Declared In `Window.h`

Prototype

```
void WinPaintRectangle (const RectangleType *rP,
    UInt16 cornerDiam)
```

Parameters

→ *rP*
Pointer to the rectangle to draw.

→ *cornerDiam*
Radius of rounded corners. Specify zero for square corners.

Returns Returns nothing.

Comments The `cornerDiam` parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides.

This function uses the current drawing state, which is stored in a `DrawStateType` structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinDrawRectangle(), WinEraseRectangle(), WinFillRectangle(), WinInvertRectangle()

WinPaintRectangleFrame Function

Purpose Draw a rectangular frame in the draw window using the current drawing state.

Declared In `Window.h`

Prototype

```
void WinPaintRectangleFrame (FrameType frame,
    const RectangleType *rP)
```

Parameters

→ *frame*
Type of frame to draw (see `FrameType`).

→ *rP*
Pointer to the rectangle to frame.

Returns Returns nothing.

Comments The frame is drawn outside the specified rectangle.

This function uses the current drawing state, which is stored in a `DrawStateType` structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinDrawGrayRectangleFrame(), WinDrawRectangleFrame(), WinEraseRectangleFrame(), WinGetFramesRectangle(), WinInvertRectangleFrame(), WinPaintRoundedRectangleFrame()

WinPaintRoundedRectangleFrame Function

Purpose Draw a rectangular frame with rounded corners in the draw window using the current drawing state.

Declared In `Window.h`

Prototype

```
void WinPaintRoundedRectangleFrame
    (const RectangleType *rP, Coord width,
    Coord cornerRadius, Coord shadowWidth)
```

Parameters

→ *rP*
Pointer to the rectangle to frame.

→ *width*
The width of the frame, interpreted using the active coordinate system.

→ *cornerRadius*
The radius of the rectangle's rounded corners, interpreted using the active coordinate system.

→ *shadowWidth*
The shadow offset, interpreted using the active coordinate system.

Returns Returns nothing.

Comments This function allows you to draw a rectangle with a frame width and corner radius specified in the active coordinate system. It is necessary because `WinPaintRectangleFrame()` doesn't allow you to draw rounded rectangles with a frame width greater than 2. Note that because there isn't a function that parallels either `WinDrawRectangleFrame()`, `WinEraseRectangleFrame()`, or `WinInvertRectangleFrame()`, you must set the drawing mode and colors as appropriate and use `WinPaintRoundedRectangleFrame` to achieve the desired effect.

Compatibility Implemented only if the High-Density Display Feature Set is present.

WinPaintTiledBitmap Function

Purpose Fill a rectangle with a pattern defined by a bitmap.

Declared In `Window.h`

Prototype

```
void WinPaintTiledBitmap (BitmapType *bitmapP,
   RectangleType *rectP)
```

Parameters

→ *bitmapP*
Pointer to the bitmap that contains the desired pattern.

→ *rectP*
Pointer to the rectangle that is to be filled.

Returns Returns nothing. On a debug ROM, if either `bitmapP` or `rectP` are `NULL`, an error is displayed.

Comments This function makes it possible for an application to define a pattern that is larger than the standard 8 by 8 custom pattern, and to define high-density custom patterns.

The pattern is scaled by the blitter using the density of `bitmapP` and the density of the screen bitmap. `bitmapP` can be a bitmap family; if it is, the Window Manager selects a bitmap using the same algorithm used by `WinPaintBitmap()`. As with other patterns, the tiled pattern is anchored to the window's origin.

If `bitmapP` does not match the depth or density of the destination bitmap, the blitter converts the bitmap using a temporary buffer. Note that if there isn't enough heap space for the temporary buffer, `WinPaintTiledBitmap` will be slow.

Compatibility Implemented only if the High-Density Display Feature Set is present.

WinPalette Function

Purpose Set or retrieve the palette for the draw window.

Declared In `Window.h`

Prototype

```
Err WinPalette (UInt8 operation, Int16 startIndex,
   UInt16 paletteEntries, RGBColorType *tableP)
```

Parameters

→ *operation*
Specify one of the following values:

`winPaletteGet`
: Retrieve the palette. Entries are read from the palette beginning at `startIndex` and placed into `tableP` beginning at index 0.

`winPaletteSet`
: Set the palette. Entries from `tableP` (beginning at index 0) are set into the palette beginning at `startIndex` in the palette.

`winPaletteSetToDefault`
: Set the palette to the default system palette.

→ *`startIndex`*
: Identifies where in the palette to start reading or writing. Specify `WinUseTableIndexes` to indicate that the entries are not to be set or read sequentially; instead, the `index` value in each `RGBColorType` entry in `tableP` determines which slot in the palette is to be set or read. You can use this technique to get or set several discontiguous palette entries with a single function call.

→ *`paletteEntries`*
: Number of palette entries to get or set.

↔ *`tableP`*
: A pointer to a buffer of `RGBColorType` entries that is either read from or written to, depending on the `operation` parameter; the table entries from 0 to `paletteEntries` – 1 are affected by this routine.

Returns Returns one of the following values:

`errNone`
: Success.

`winErrPalette`
: The current draw window does not have a color table, a set operation has overflowed the color table, or one of the entries in `tableP` has an invalid index value

`sysErrParamErr`
: The `startIndex` value is invalid.

Comments Here are some examples of how this routine works:

- If `startIndex` is 0 and `paletteEntries` is 10, the first 10 elements of the palette will be set from `tableP` or will be copied into `tableP`.
- If `startIndex` is 10 and `paletteEntries` is 5, then entries 10, 11, 12, 13, and 14 in the palette will be set from or copied to elements 0, 1, 2, 3, and 4 in `tableP`.
- If `startIndex` is `WinUseTableIndexes` and `paletteEntries` is 1, then the index value in the `RGBColorType` of element 0 of `tableP` will be read from or copied to `tableP`; in this case, the `index` field of the `RGBColorType` will not change.

During a set operation, this function broadcasts the `sysNotifyDisplayChangeEvent` to notify any interested observer that the color palette has changed.

One use for this function is if you need to display a bitmap that uses a color table other than the one in use by the system. You can attach a custom color table to a bitmap, and if you do, the bitmap is drawn using that color table. However, this is a performance drain. As an optimization, you can use `WinPalette` to change the system color table to match that used by the bitmap, display the bitmap, and use `WinPalette` to reset the color table when the bitmap is no longer visible.

Compatibility Implemented only if 3.5 New Feature Set is present.

WinPopDrawState Function

Purpose Restore the draw state values to the last saved set on the stack.

Declared In `Window.h`

Prototype

```
void WinPopDrawState (void)
```

Parameters `None.`

Returns Returns nothing.

Comments Use this routine to restore the draw state saved by the previous call to `WinPushDrawState()`.

After you call this function, the current draw window's `drawStateP` field points to the restored drawing state.

Compatibility Implemented only if 3.5 New Feature Set is present.

WinPushDrawState Function

Purpose Save the current draw state values onto the draw state stack.

Declared In `Window.h`

Prototype

```
void WinPushDrawState (void)
```

Parameters `None.`

Returns Returns nothing.

Comments Use this routine to save the current draw state before making changes to it using the functions listed in the `DrawStateType` structure's description. Call `WinPopDrawState()` to restore the saved settings.

Compatibility Implemented only if 3.5 New Feature Set is present.

WinResetClip Function

Purpose Reset the clipping rectangle of the draw window to the portion of the draw window that is within the bounds of the display.

Declared In `Window.h`

Prototype

```
void WinResetClip (void)
```

Parameters `None.`

Returns Returns nothing.

See Also WinSetClip()

WinRestoreBits Function

Purpose Copy the contents of the specified window to the draw window and delete the passed window.

Declared In `Window.h`

Prototype

```
void WinRestoreBits (WinHandle winHandle,
    Coord destX, Coord destY)
```

Parameters

→ *winHandle*
Handle of window to copy and delete.

→ *destX*
x coordinate in the draw window to copy to.

→ *destY*
y coordinate in the draw window to copy to.

Returns Returns nothing.

Comments This routine is generally used to restore a region of the display that was saved with `WinSaveBit()`.

See Also WinSaveBit()

WinRGBToIndex Function

Purpose Convert an RGB value to the index of the closest color in the currently active color lookup table (CLUT).

Declared In `Window.h`

Prototype

```
IndexedColorType WinRGBToIndex
    (const RGBColorType *rgbP)
```

Parameters

→ *rgbP*
Pointer to an RGB color value.

Returns Returns the index of the closest matching color in the CLUT.

Comments Palm OS 3.5 supports a maximum of 256 colors. The number of possible RGB colors greatly exceeds this amount. For this reason, an exact match may not be available for `rgbP`. If there is no exact RGB match, then a luminance best-fit is used if the color lookup table is entirely gray scale (red, green, and blue values for each entry are identical), or a shortest-distance fit in RGB space is used if the palette contains colors. RGB shortest distance may not always

produce the actual closest perceptible color, but it's relatively fast and works for the system palette.

`WinRGBToIndex` uses the draw window's color table to return the appropriate color table index. If the draw window does not have a color table, the default color table of the current screen is used.

If the draw window does not have a color table, and if the depth of the draw window and the depth of the screen are different, this function will return an inappropriate index. If this situation exists, the application should either define a color table for the draw window, or use `WinScreenMode()` to set the screen depth to the same depth as the draw window before calling `WinRGBToIndex`.

NOTE: The bitmap data will not be blitted properly if the depth of the screen is changed using `WinScreenMode()` and the new window uses a bitmap that does not define the bitmap's color table. See `WinScreenMode()` for information on how to work around this limitation.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `WinIndexToRGB()`, `WinScreenMode()`

WinSaveBit Function

Purpose Create an offscreen window and copy the specified region from the draw window to the offscreen window.

Declared In `Window.h`

Prototype

```
WinHandle WinSaveBits
    (const RectangleType *source, UInt16 *error)
```

Parameters

→ *source*
Pointer to the bounds of the region to save, relative to the display.

← *error*
Pointer to any error encountered by this function.

Returns Returns the handle of the window containing the saved image, or zero if an error occurred.

Comments The offscreen window is the same size as the region to copy.

This function tries to copy the window's bitmap using compressed format if possible. It may display a fatal error message if an error occurs when it tries to shrink the pointer for the compressed bits.

See Also WinRestoreBits()

WinScaleCoord Function

Purpose Convert a single coordinate from the standard coordinate system to the active coordinate system.

Declared In `Window.h`

Prototype

```
Coord WinScaleCoord (Coord coord, Boolean ceiling)
```

Parameters

→ *coord*
A coordinate in the standard coordinate system.

→ *ceiling*
Pass `true` to round up, `false` to truncate the fractional part when scaling.

Returns Returns the coordinate scaled to the active coordinate system.

Comments This function converts a coordinate by multiplying it by the coordinate scaling factor, and then truncating or rounding the result to an integer value depending on the value of `ceiling`.

If the active coordinate system is `kCoordinatesStandard`, the returned coordinate is equal to the supplied coordinate.

Compatibility Implemented only if the High-Density Display Feature Set is present.

On Palm OS Cobalt, the scaling algorithm is slightly different. After the coordinate scale factor is applied, what happens next depends upon the value of *ceiling*:

if *ceiling* == true {

use `lfloorf()` function when scaleFactor > 1

use `lceilf()` function when scaleFactor < 1

else

use `lceilf()` function when scaleFactor > 1

use `lfloorf()` function when scaleFactor < 1

}

The objective of this algorithm is to ensure that for any `Coord`, `Point`, or `Rectangle` "x", x = unscaled(scaled(x)).

See Also `WinScalePoint()`, `WinScaleRectangle()`, `WinUnscaleCoord()`

WinScalePoint Function

Purpose Convert a point from the standard coordinate system to the active coordinate system.

Declared In `Window.h`

Prototype

```
void WinScalePoint (PointType *pointP,
    Boolean ceiling)
```

Parameters

↔ *pointP*
: Pointer to a `PointType` structure that, before the call, should contain a point's standard coordinate system coordinates. After this function is called the `PointType` structure contains the coordinates of the point scaled to the active coordinate system.

→ *ceiling*
: Pass `true` to round up, `false` to truncate the fractional part when scaling.

Returns Returns nothing. The coordinates of the point indicated by `pointP` are converted to the active coordinate system.

Comments This function converts a point by multiplying its x and y coordinates by the coordinate scaling factor and then truncating or rounding the results depending on the value of `ceiling`.

If the active coordinate system is `kCoordinatesStandard`, `pointP` is not changed by this function.

Compatibility Implemented only if the High-Density Display Feature Set is present.

On Palm OS Cobalt, the scaling algorithm is slightly different. After the coordinate scale factor is applied, what happens next depends upon the value of *ceiling*:

if *ceiling* == true {

use `lfloorf()` function when scaleFactor > 1

use `lceilf()` function when scaleFactor < 1

else

use `lceilf()` function when scaleFactor > 1

use `lfloorf()` function when scaleFactor < 1

}

The objective of this algorithm is to ensure that for any `Coord`, `Point`, or `Rectangle` "x", x = unscaled(scaled(x)).

See Also `WinScaleCoord()`, `WinScaleRectangle()`, `WinUnscalePoint()`

WinScaleRectangle Function

Purpose Convert a rectangle from the standard coordinate system to the active coordinate system.

Declared In `Window.h`

Prototype

```
void WinScaleRectangle (RectangleType *rectP)
```

Parameters

↔ *rectP*
: Pointer to a `RectangleType` structure that, before the call, should contain a rectangle's standard coordinate system coordinates. After this function is called the `RectangleType` structure contains the coordinates of the rectangle scaled to the active coordinate system.

Returns Returns nothing. The coordinates of the rectangle indicated by `rectP` are converted to the native coordinate system.

Comments This function scales the rectangle's `topLeft` and `extent` points by multiplying their `x` and `y` coordinates by the coordinate scaling factor. All values are then truncated, but if either `topLeft.x` or `extent.x` had a fractional part, `extent.x` is incremented by 1 (and, similarly, if either `topLeft.y` or `extent.y` had a fractional part, `extent.y` is incremented by 1).

If the active coordinate system is `kCoordinatesStandard`, `rectP` is not changed by this function.

You can use this function when your gadget handler draws using a more precise coordinate system than the Form Manager and needs to convert the form-based bounds of the gadget to the high-density bounds used by the gadget's drawing function.

Compatibility Implemented only if the High-Density Display Feature Set is present.

See Also `WinScaleCoord()`, `WinScalePoint()`, `WinUnscaleRectangle()`

WinScreenGetAttribute Function

Purpose Get various attributes of the screen.

Declared In `Window.h`

Prototype

```
Err WinScreenGetAttribute
    (WinScreenAttrType selector, UInt32 *attrP)
```

Parameters

→ *selector*
: A value indicating which attribute to return. See the description of `WinScreenAttrType` in the Comments section, below, for the values you can supply to this parameter.

← *attrP*
: Pointer to a `UInt32` into which the specified attribute value is placed by this function.

Returns Returns `errNone` if the function successfully retrieved the specified attribute, or `sysErrParamErr` if `selector` doesn't represent a screen attribute.

Comments This function returns many of the attributes that can be obtained with `WinScreenMode()`. Unlike `WinScreenMode`, however, this function can also return the number of bytes used by each row in the screen buffer as well as the number of pixels per inch on the screen's x and y axes.

Unlike `WinScreenMode`, you cannot set any attributes with this function. Also, you cannot use this function to obtain the "color enabled" attribute. And unlike `WinScreenMode`, this function always returns the true screen dimensions; `WinScreenMode` converts the dimensions to the active coordinate system.

Applications can use the screen resolution information to make intelligent decisions about how to draw primitives on Palm Powered handhelds with different screen resolutions.

WinScreenAttrType

This enum defines the selectors that can be used with the `WinScreenGetAttribute()` function.

Prototype

```
typedef enum {
  winScreenWidth,
  winScreenHeight,
  winScreenRowBytes,
  winScreenDepth,
  winScreenAllDepths,
  winScreenDensity,
  winScreenPixelFormat,
  winScreenResolutionX,
  winScreenResolutionY
} WinScreenAttrType;
```

Constants

`winScreenWidth`
: The width of the screen, in pixels.

`winScreenHeight`
: The height of the screen, in pixels.

`winScreenRowBytes`
: The number of bytes used by each row in the screen buffer.

`winScreenDepth`
: The screen depth.

`winScreenAllDepths`
: All screen depths (in bitmap format).

`winScreenDensity`
: The screen bitmap's density.

`winScreenPixelFormat`
: The `PixelFormatType` appropriate for the screen.

`winScreenResolutionX`
: The number of pixels per inch along the screen's x axis.

`winScreenResolutionY`
: The number of pixels per inch along the screen's y axis.

Compatibility Implemented only if the High-Density Display Feature Set is present.

See Also `WinScreenMode()`

WinScreenLock Function

Purpose "Lock" the current screen by switching the UI concept of the screen base address to an area that is not reflected on the display.

Declared In `Window.h`

Prototype

```
UInt8 *WinScreenLock (WinLockInitType initMode)
```

Parameters → *`initMode`*

Indicates how to initialize the new screen area. Specify one of the following values:

`winLockCopy`
: Copy old screen to new.

`winLockErase`
: Erase new screen to white.

`winLockDontCare`
: Don't do anything

Returns Returns a pointer to the new screen base address, or `NULL` if this routine fails.

Comments This routine can be used to "freeze" the display while doing lengthy drawing operations to avoid a flickering effect. Call `WinScreenUnlock()` to unlock the display and cause it to be updated with any changes. The screen must be unlocked as many times as it is locked to actually update the display.

Because this function copies the screen, using it is a relatively expensive operation.

Compatibility Implemented only if 3.5 New Feature Set is present.

WinScreenMode Function

Purpose Sets or returns display parameters, including display geometry, bit depth, and color support.

Declared In `Window.h`

Prototype

```
Err WinScreenMode
    (WinScreenModeOperation operation,
    UInt32 *widthP, UInt32 *heightP,
    UInt32 *depthP, Boolean *enableColorP)
```

Parameters The `widthP`, `heightP`, `depthP`, and `enableColorP` parameters are used in different ways for different operations. See Comments at the end of this description for details.

→ *`operation`*

The work this function is to perform, as specified by one of the following selectors:

`winScreenModeGet`
: Return the current settings for the display.

`winScreenModeGetDefaults`
: Return the default settings for the display.

`winScreenModeGetSupportedDepths`
: Return in `depthP` a hexadecimal value indicating the supported screen depths. The binary representation of this value defines a bitfield in which the value 1 indicates support for a particular display depth. The position representing a particular bit depth corresponds to the value $2^{(bitDepth-1)}$. See the Example at the end of this function description for more information.

`winScreenModeGetSupportsColor`
: Return `true` as the value of the `enableColorP` parameter when color mode can be enabled.

`winScreenModeSet`
: Change display settings to the values specified by the other arguments to the `WinScreenMode` function.

`winScreenModeSetToDefaults`
: Change display settings to default values.

↔ *widthP*
: Pointer to new/old screen width. For backward compatibility, when `operation` is `winScreenModeGet` or `winScreenModeGetDefaults`, a single-density width is returned, even if the handheld has a double-density display. Use `WinScreenGetAttribute()` to retrieve the true hardware dimensions of the display.

↔ *heightP*
: Pointer to new/old screen height. For backward compatibility, when `operation` is `winScreenModeGet` or `winScreenModeGetDefaults`, a single-density height is returned, even if the handheld has a double-density display. Use `WinScreenGetAttribute()` to retrieve the true hardware dimensions of the display.

↔ *depthP*
: Pointer to new/old/available screen depth.

↔ *enableColorP*
: Pass `true` to enable color drawing mode. The returned value (when using an operation that returns a value through this parameter) simply indicates whether or not the hardware supports color; its value does not change based on the current screen depth.

Returns

If no error, returns values as specified by the `operation` argument. Various invalid arguments may cause this function to return a `sysErrParamErr` result code. In rare cases, a failed allocation can cause this function to return a `memErrNotEnoughSpace` error.

Comments

The `widthP`, `heightP`, `depthP`, and `enableColorP` parameters are used in different ways for different operations. All "get" operations overwrite these values with a result when the function returns. The `winScreenModeSet` operation changes current display parameters when passed valid argument values that are not `NULL` pointers. The `winScreenModeSetToDefaults` operation ignores values passed for all of these parameters.

Table 59.1 summarizes parameter usage for each operation this function performs.

Table 59.1 Use of parameters to WinScreenMode function

Operation `winScreenMode...`	widthP	heightP	depthP	enableColorP
`...Get`	returned	returned	returned	returned
`...GetDefaults`	returned	returned	returned	returned
`...GetSupportedDepths`	ignored	ignored	returned	pass in
`...GetSupportsColor`	ignored	ignored	pass in	returned
`...Set`	pass in	pass in	pass in	pass in
`...SetToDefaults`	ignored	ignored	ignored	ignored

This function ignores `NULL` pointer arguments to the `widthP`, `heightP`, `depthP`, and `enableColorP` parameters; thus, you can pass a `NULL` pointer for any of these values to leave the current value unchanged. Similarly, when getting values, this function does not return a value for any `NULL` pointer argument.

If you change the display depth, it is recommended that you restore it to its previous state when your application closes, even though the system sets display parameters back to their default values when launching an application.

Note that none of the other operations interprets the depth parameter the same way that `winScreenModeGetSupportedDepths` does. For example, to set the display depth to 8-bit mode, you use 8 (decimal) for the display depth, not 0x80 (128 decimal).

When a window is created, and if the window's associated bitmap does not have its own color table, the window will use the system's default color translation tables when a blitting operation occurs to that window. When the system's bit depth changes, the system's default color translation tables are recalculated based on the new screen depth. When the blit occurs at the new screen depth to the offscreen window, the color translation tables are out of sync.

To work around this system limitation, developers should either:

- allocate offscreen windows after changing the depth, or

- use `WinCreateBitmapWindow()` so that it uses a bitmap with a defined color table.

The latter workaround causes the system to perform color matching when blitting, so the first workaround may be preferred.

Compatibility Implemented only if 3.5 New Feature Set is present. In OS versions prior to 3.5, this function is called `ScrDisplayMode`. The prototype for `ScrDisplayMode` is similar to `WinScreenMode`:

```
Err ScrDisplayMode ( ScrDisplayModeOperation operation,
DWordPtr widthP, DWordPtr heightP,
DWordPtr depthP, BooleanPtr enableColorP)
```

The only other difference between `ScrDisplayMode` and `WinScreenMode` is that the `ScrDisplayModeOperation` constants begin with the prefix `scrDisplayMode` rather than `winScreenMode`.

Example Here are some additional examples of return values provided by the `winScreenModeGetSupportedDepths` mode of the `WinScreenMode` function.

This function indicates support for 4-bit drawing by returning a value of `0x08`, or 2^3, which corresponds to a binary value of `1000`. Support for bit depths of 2 and 1 is indicated by a return value of `0x03`. Support for bit depths of 4, 2, and 1 is indicated by `0x0B`, which is a binary value of `1011`. Support for bit depths of 16, 8, 4 and 2 is indicated by `0x808A`. The figure immediately following depicts this final example graphically.

See Also `WinScreenGetAttribute()`

WinScreenUnlock Function

Purpose Unlock the screen and update the display.

Declared In `Window.h`

Prototype

```
void WinScreenUnlock (void)
```

Parameters `None.`

Returns Returns nothing.

Comments The screen must be unlocked as many times as it is locked to actually update the display.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinScreenLock()

WinScrollRectangle Function

Purpose Scroll a rectangle in the draw window.

Declared In `Window.h`

Prototype

```
void WinScrollRectangle (const RectangleType *rP,
   WinDirectionType direction, Coord distance,
   RectangleType *vacatedP)
```

Parameters

→ *rP*
Pointer to the rectangle to scroll.

→ *direction*
Direction to scroll (`winUp`, `winDown`, `winLeft`, or `winRight`).

→ *distance*
Distance to scroll in pixels.

← *vacatedP*
Pointer to the rectangle that needs to be redrawn because it has been vacated as a result of the scroll.

Returns Returns nothing.

Comments The rectangle scrolls within its own bounds. Any portion of the rectangle that is scrolled outside its bounds is clipped.

WinSetActiveWindow Function

Purpose Make a window the active window.

Declared In `Window.h`

Prototype
```
void WinSetActiveWindow (WinHandle winHandle)
```

Parameters → *winHandle*
Handle of a window.

Returns Returns nothing.

Comments The active window is not actually set in this routine; flags are set to indicate that a window is being exited and another window is being entered. The routine `EvtGetEvent` sends a `winExitEvent` and a `winEnterEvent` when it detects these flags. The active window is set by `EvtGetEvent` when it sends the `winEnterEvent`. The draw window is also set to the new active window when the active window is changed.

The window is enabled before it is made active.

All user input is directed to the active window.

See Also `WinGetActiveWindow()`, `EvtGetEvent()`

WinSetBackColor Function

Purpose Set the background color to use in subsequent draw operations.

Declared In `Window.h`

Prototype
```
IndexedColorType WinSetBackColor
    (IndexedColorType backColor)
```

Parameters → *backColor*
Color to set; specify a value of type `IndexedColorType`.

Returns Returns the previous background color index.

Comments This function changes the current drawing state. If necessary, use `WinPushDrawState()` to preserve the current drawing state before you set this function and use `WinPopDrawState()` to restore it later.

To set the foreground color to a predefined UI color default, use UIColorGetTableEntryIndex() as an input to this function. For example:

```
curColor = WinSetBackColor
(UIColorGetTableEntryIndex(UIFieldBackground));
```

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinSetForeColor(), WinSetTextColor()

WinSetBackColorRGB Function

Purpose Set the background color to use in subsequent draw operations.

Declared In `Window.h`

Prototype

```
void WinSetBackColorRGB
    (const RGBColorType *newRgbP,
    RGBColorType *prevRgbP)
```

Parameters

→ *newRgbP*
Color to set; specify a value of type RGBColorType.

← *prevRgbP*
Previous color; specify a value of type RGBColorType.

Returns Returns nothing

Comments This function takes new and previous RGBColorType arguments. It is okay to set `newRgbP` or `prevRgbP` to `NULL`. If an application only wants to get the current color, the `newRgbP` argument is set to `NULL`. If the application does not care about the previous color, `prevRgbP` can be set to `NULL`.

This function sets the `backColorRGB` field of the DrawStateType structure to the value specified by `newRgbP`. It then sets the index field of `backColorRGB` to the 8 bit system palette entry that most closely matches the RGB components. Finally, it sets the `backColor` index field of DrawStateType to this index value.

This function changes the current drawing state. If necessary, use WinPushDrawState() to preserve the current drawing state before you set this function and use WinPopDrawState() to restore it later.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also WinSetForeColorRGB(), WinSetTextColorRGB()

WinSetBounds Function

Purpose Set the bounds of the window to display-relative coordinates.

Declared In `Window.h`

Prototype

```
void WinSetBounds (WinHandle winHandle,
    const RectangleType *rP)
```

Parameters

→ *winHandle*
Handle for the window for which to set the bounds.

→ *rP*
Pointer to a rectangle to use for bounds.

Returns `Returns nothing.`

Comments A visible window cannot have its bounds modified.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also WinGetBounds()

WinSetClip Function

Purpose Set the clipping rectangle of the draw window.

Declared In `Window.h`

Prototype

```
void WinSetClip (const RectangleType *rP)
```

Parameters

→ *rP*
Pointer to a structure holding the clipping bounds.

Returns Returns nothing.

See Also WinClipRectangle(), WinSetClip(), WinGetClip()

WinSetConstraintsSize Function

Purpose Sets the maximum, preferred, and minimum size constraints for a window.

Declared In `Window.h`

Prototype

```
Err WinSetConstraintsSize (WinHandle winHandle,
   Coord minH, Coord prefH, Coord maxH, Coord minW,
   Coord prefW, Coord maxW)
```

Parameters

→ *winHandle*
: A handle to a window.

→ *minH*
: The minimum height to which this window can be sized in standard coordinates. This value must be less than or equal to 160 pixels.

→ *prefH*
: The preferred height for this window in standard coordinates.

→ *maxH*
: The maximum height for this window in standard coordinates.

→ *minW*
: The minimum width for the window in standard coordinates.

→ *prefW*
: The preferred width for the window in standard coordinates.

→ *maxW*
: The maximum width for the window in standard coordinates.

Returns Returns one of the following error codes:

`errNone`
: Success.

`pinErrNoSoftInputArea`
: There is no dynamic input area on this device.

`pinErrInvalidParam`
: You have specified an invalid parameter.

Comments Applications must call this function in response to a `frmLoadEvent` or `frmOpenEvent` to set the size constraints for the window. If the application does not call this function, it is assumed that both the minimum and maximum values for the window are 160 pixels by 160 pixels.

Compatibility Implemented only if the Pen Input Manager Feature Set is present.

WinSetCoordinateSystem Function

Purpose Establish the coordinate system to be used for subsequent drawing operations.

Declared In `Window.h`

Prototype

```
UInt16 WinSetCoordinateSystem (UInt16 coordSys)
```

Parameters → *coordSys*

The desired coordinate system. Supply one of the values defined in "Window Coordinate System Constants" on page 1227.

Returns Returns the previous coordinate system value.

Comments This function modifies the `scale` field in the draw state (a `DrawStateType` structure). As when making other modifications to a window's draw state, applications should call `WinPushDrawState()` before modifying the coordinate system. To restore the coordinate system, your application can then call `WinPopDrawState()`.

To calculate the draw state `scale` field, the Window Manager divides the density of the bitmap associated with the draw window by `coordSys`. If `coordSys` is `kCoordinatesNative`, the Window Manager sets the scale field to 1.0, which to enables 1-to-1 mapping of coordinates to pixels.

If you supply a value of `kCoordinatesStandard` for `coordSys`, subsequent drawing will use the standard coordinate system.

Compatibility Implemented only if the High-Density Display Feature Set is present.

See Also `WinGetCoordinateSystem()`

WinSetDrawMode Function

Purpose Set the transfer mode to use in subsequent draw operations.

Declared In `Window.h`

Prototype

```
WinDrawOperation WinSetDrawMode
    (WinDrawOperation newMode)
```

Parameters → *newMode*
Transfer mode to set; specify one of the `WinDrawOperation` values.

Returns Returns the previous transfer mode.

Comments This function changes the current drawing state. If necessary, use `WinPushDrawState()` to preserve the current drawing state before you set this function and use `WinPopDrawState()` to restore it later.

Compatibility Implemented only if 3.5 New Feature Set is present.

WinSetDrawWindow Function

Purpose Set the draw window. (All drawing operations are relative to the draw window.)

Declared In `Window.h`

Prototype

```
WinHandle WinSetDrawWindow (WinHandle winHandle)
```

Parameters → *winHandle*
Handle of a window.

Returns Returns the previous draw window.

Compatibility OS versions before 3.5 allowed you to use `NULL` as a parameter to this function to set the draw window to the display window (or screen window). In version 3.5 and higher, this practice is discouraged. If `winHandle` is `NULL`, the debug ROM sets the draw window to `badDrawWindowValue`, and you are warned if you try to draw to it.

See Also WinGetDrawWindow(), WinSetActiveWindow()

WinSetForeColor Function

Purpose Set the foreground color to use in subsequent draw operations.

Declared In `Window.h`

Prototype

```
IndexedColorType WinSetForeColor
    (IndexedColorType foreColor)
```

Parameters → *foreColor*
Color to set; specify a value of type `IndexedColorType`.

Returns Returns the previous foreground color index.

Comments This function changes the current drawing state. If necessary, use `WinPushDrawState()` to preserve the current drawing state before you set this function and use `WinPopDrawState()` to restore it later.

To set the foreground color to a predefined UI color default, use `UIColorGetTableEntryIndex()` as an input to this function. For example:

```
curColor = WinSetForeColor
  (UIColorGetTableEntryIndex
  (UIObjectForeground));
```

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinSetBackColor(), WinSetTextColor()

WinSetForeColorRGB Function

Purpose Set the foreground color to use in subsequent draw operations.

Declared In `Window.h`

Prototype

```
void WinSetForeColorRGB
    (const RGBColorType *newRgbP,
    RGBColorType *prevRgbP)
```

Parameters → *newRgbP*
Color to set; specify a value of type `RGBColorType`.

← *prevRgbP*
Previous color; specify a value of type `RGBColorType`.

Returns Returns nothing.

Comments This function takes new and previous `RGBColorType` arguments. It is okay to set `newRgbP` or `prevRgbP` to `NULL`. If an application only wants to get the current color, the `newRgbP` argument is set to `NULL`. If the application does not care about the previous color, `prevRgbP` can be set to `NULL`.

This function sets the `foreColorRGB` field of the `DrawStateType` structure to the value specified by `newRgbP`. It then sets the index field of `foreColorRGB` to the 8 bit system palette entry that most closely matches the RGB components. Finally, it sets the `foreColor` index field of `DrawStateType` to this index value.

This function changes the current drawing state. If necessary, use `WinPushDrawState()` to preserve the current drawing state before you set this function and use `WinPopDrawState()` to restore it later.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also WinSetBackColorRGB(), WinSetTextColorRGB()

WinSetPattern Function

Purpose Set the current fill pattern.

Declared In `Window.h`

Prototype

```
void WinSetPattern
    (const CustomPatternType *patternP)
```

Parameters → *patternP*
Pattern to set (see `CustomPatternType`).

Returns Returns nothing.

Comments The fill pattern is used by `WinFillLine()` and `WinFillRectangle()`.

This function changes the current drawing state. If necessary, use `WinPushDrawState()` to preserve the current drawing state before you set this function and use `WinPopDrawState()` to restore it later.

See Also WinGetPattern()

WinSetPatternType Function

Purpose Set the current pattern type.

Declared In `Window.h`

Prototype
```
void WinSetPatternType (PatternType newPattern)
```

Parameters → *newPattern*
Pattern type to set for the draw window (see `PatternType`).

Returns Returns nothing.

Comments This function sets the `pattern` field of the drawing state to `newPattern` and sets the `patternData` field to `NULL`. To set `patternData` to a custom pattern use `WinSetPattern()`.

The fill pattern is used by `WinFillLine()` and `WinFillRectangle()`.

This function changes the current drawing state. If necessary, use `WinPushDrawState()` to preserve the current drawing state before you set this function and use `WinPopDrawState()` to restore it later.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinGetPatternType()

WinSetScalingMode Function

Purpose Controls the scaling of bitmaps and fonts, and the spacing between text glyphs.

Declared In `Window.h`

Prototype
```
UInt32 WinSetScalingMode (UInt32 mode)
```

Parameters → *mode*
Specifies whether bitmaps and fonts should be scaled, and whether text glyphs should be aligned on standard coordinates. See the Comments section, below, for details on how to construct the `mode` value.

Returns Returns the previous scaling mode.

Comments If you need to display a large, low-density bitmap—for instance, a map or a photograph—you can use `WinSetScalingMode` to draw

your bitmap unscaled, allowing the user to see more of the bitmap at one time. If you want to draw more, but smaller, text on a handheld that has a high-density display (including 1.5 and double-density), you can use `WinSetScalingMode` to draw text unscaled. Finally, when the window coordinate system is set to `kCoordinatesOneAndAHalf` you can use this function to draw text without padding pixels between glyphs, resulting in more consistent kerning for text.

On a handheld with a high-density display, if the bitmap being drawn, or the font being used, is part of a family that contains both low- and high-density members, `WinSetScalingMode` controls which member is used. If `kTextScalingOff` is set, for instance, the low-density font is used, unscaled, when drawing subsequent text.

Scaling Mode Values

Depending on how bitmaps and text should be scaled, OR the appropriate constants from the following table to construct the appropriate scaling mode value.

Constant	Value	Description
`kBitmapScalingOff`	1	Turning bitmap scaling off causes the blitter to draw the low-density bitmap family member, unscaled, when the application subsequently calls `WinDrawBitmap()` or `WinPaintBitmap()` (even if the bitmap family contains a bitmap who's density matches the window's). This causes your bitmaps to be drawn smaller, as if they were being viewed from a distance. The application uses the coordinates passed to `WinDrawBitmap` and `WinCopyRectangle()` to determine where the unscaled low-density bitmap is placed within the larger footprint of the high-density bounds.
`kTextScalingOff`	2	Turning text scaling off causes the blitter to render text using the low-density family member of the appropriate font, unscaled, when the application subsequently draws text (even if the font family contains a font who's density matches the window's). This results in smaller-sized text being drawn on the screen.
`kTextPaddingOff`	4	When text padding is turned off, the blitter does not insert pixels between glyphs in order to align glyphs on standard coordinates. (Text padding is on by default.) By setting the `kTextPaddingOff` bit, an application can lay out text using high density coordinates and preserve the native font's internal kerning. This unpadded mode provides the best appearance of text, but requires the application to use the high-density coordinate system.

When an application calls `WinSetCoordinateSystem()` to use native coordinates for a high-density window, the operating system turns off text padding. When an application calls `WinSetCoordinateSystem` to use standard coordinates, the operating system turns text padding on. To use standard coordinates with pixel padding turned off, or to use native coordinates with pixel padding turned on, call

`WinSetScalingMode`aftercalling`WinSetCoordinateSystem()`. Note that using standard coordinates with padding off is not recommended since the Font Manager API that measures character widths will return inaccurate widths.

The scaling mode flags are part of the window's draw state. In order to ensure that the draw state is properly restored, save that state using `WinPushDrawState()` before calling functions that alter the draw state—such as `WinSetScalingMode` or `WinSetCoordinateSystem`. After drawing is completed, restore the draw state with `WinPopDrawState()`.

When an application quits, the draw state is restored to the default settings.

Compatibility Implemented only if the 1.5x Display Feature Set is present.

See Also `WinGetScalingMode()`

WinSetTextColor Function

Purpose Set the color to use for drawing characters in subsequent draw operations.

Declared In `Window.h`

Prototype

```
IndexedColorType WinSetTextColor
    (IndexedColorType textColor)
```

Parameters

→ *textColor*
Color to set; specify a value of type `IndexedColorType`.

Returns Returns the previous text color index.

Comments This function changes the current drawing state. If necessary, use `WinPushDrawState()` to preserve the current drawing state before you set this function and use `WinPopDrawState()` to restore it later.

To set the foreground color to a predefined UI color default, use `UIColorGetTableEntryIndex()` as an input to this function. For example:

```
curColor = WinSetTextColor
  (UIColorGetTableEntryIndex(UIFieldText));
```

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also WinSetBackColor(), WinSetForeColor()

WinSetTextColorRGB Function

Purpose Set the color to use for drawing characters in subsequent draw operations.

Declared In `Window.h`

Prototype

```
void WinSetTextColorRGB
    (const RGBColorType *newRgbP,
    RGBColorType *prevRgbP)
```

Parameters

→ *newRgbP*
Color to set; specify a value of type `RGBColorType`.

← *prevRgbP*
Previous color; specify a value of type `RGBColorType`.

Returns Returns nothing.

Comments This function takes new and previous `RGBColorType` arguments. It is okay to set `newRgbP` or `prevRgbP` to `NULL`. If an application only wants to get the current color, the `newRgbP` argument is set to `NULL`. If the application does not care about the previous color, `prevRgbP` can be set to `NULL`.

This function sets the `textColorRGB` field of the `DrawStateType` structure to the value specified by `newRgbP`. It then sets the index field of `textColorRGB` to the 8 bit system palette entry that most closely matches the RGB components. Finally, it sets the `textColor` index field of `DrawStateType` to this index value.

This function changes the current drawing state. If necessary, use `WinPushDrawState()` to preserve the current drawing state before you set this function and use `WinPopDrawState()` to restore it later.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also WinSetBackColorRGB(), WinSetForeColorRGB()

WinSetUnderlineMode Function

Purpose Set the graphic state to enable or disable the underlining of characters.

Declared In `Window.h`

Prototype

```
UnderlineModeType WinSetUnderlineMode
    (UnderlineModeType mode)
```

Parameters

↔ *mode*
New underline mode type; see `UnderlineModeType`.

Returns Returns the previous underline mode type.

Comments This function changes the current drawing state. If necessary, use `WinPushDrawState()` to preserve the current drawing state before you set this function and use `WinPopDrawState()` to restore it later.

See Also WinDrawChars()

WinUnscaleCoord Function

Purpose Convert a single coordinate from the active coordinate system to the standard coordinate system.

Declared In `Window.h`

Prototype

```
Coord WinUnscaleCoord (Coord coord,
    Boolean ceiling)
```

Parameters

→ *coord*
A coordinate in the active coordinate system.

→ *ceiling*
Pass `true` to round up, `false` to truncate the fractional part when scaling.

Returns Returns the coordinate scaled to the standard coordinate system.

Comments This function converts a coordinate by dividing it by the coordinate scaling factor, truncating or rounding the result to an integer value depending on the value of `ceiling`.

If the active coordinate system is `kCoordinatesStandard`, the returned coordinate is equal to the supplied coordinate.

Compatibility Implemented only if the High-Density Display Feature Set is present.

On Palm OS Cobalt, the scaling algorithm is slightly different. After the coordinate scale factor is applied, what happens next depends upon the value of *ceiling*:

if *ceiling* == true {

use `lfloorf()` function when scaleFactor > 1

use `lceilf()` function when scaleFactor < 1

else

use `lceilf()` function when scaleFactor > 1

use `lfloorf()` function when scaleFactor < 1

}

The objective of this algorithm is to ensure that for any `Coord`, `Point`, or `Rectangle` "x", x = unscaled(scaled(x)).

See Also `WinScaleCoord()`, `WinUnscalePoint()`, `WinUnscaleRectangle()`

WinUnscalePoint Function

Purpose Convert a point from the active coordinate system to the standard coordinate system.

Declared In `Window.h`

Prototype

```
void WinUnscalePoint (PointType *pointP,
    Boolean ceiling)
```

Parameters

↔ *pointP*

Pointer to a `PointType` structure that, before the call, should contain a point's coordinates using the active coordinate system. After this function is called the `PointType` structure contains the coordinates of the point scaled to the standard coordinate system.

→ *ceiling*

Pass `true` to round up, `false` to truncate the fractional part when scaling.

Returns Returns nothing. The coordinates of the point indicated by `pointP` are converted to the standard coordinate system.

Comments This function converts a point by dividing its x and y coordinates by the coordinate scaling factor, truncating or rounding the results to integer values depending on the value of `ceiling`. For instance, the input coordinates (11, 13) are transformed to (6, 7) if the input values represent native coordinates on a handheld with a double-density screen and `ceiling` is set to `true`. If `ceiling` is set to `false`, the same input coordinates are transformed to (5, 6).

If the active coordinate system is `kCoordinatesStandard`, `pointP` is not changed by this function.

Compatibility Implemented only if the High-Density Display Feature Set is present.

On Palm OS Cobalt, the scaling algorithm is slightly different. After the coordinate scale factor is applied, what happens next depends upon the value of *`ceiling`*:

if *`ceiling`* == true {

  use `lfloorf()` function when scaleFactor > 1

  use `lceilf()` function when scaleFactor < 1

else

  use `lceilf()` function when scaleFactor > 1

  use `lfloorf()` function when scaleFactor < 1

}

The objective of this algorithm is to ensure that for any `Coord`, `Point`, or `Rectangle` "x", x = unscaled(scaled(x)).

See Also `WinScalePoint()`, `WinUnscaleCoord()`, `WinUnscaleRectangle()`

WinUnscaleRectangle Function

Purpose Convert a rectangle from the active coordinate system to the standard coordinate system.

Declared In `Window.h`

Prototype

```
void WinUnscaleRectangle (RectangleType *rectP)
```

Parameters ↔ *rectP*

Pointer to a `RectangleType` structure that, before the call, should contain a rectangle's coordinates using the active coordinate system. After this function is called the `RectangleType` structure contains the coordinates of the rectangle scaled to the standard coordinate system.

Returns Returns nothing. The coordinates of the rectangle indicated by `rectP` are converted to the standard coordinate system.

Comments This function scales the rectangle's `topLeft` and `extent` points by dividing their `x` and `y` coordinates by the coordinate scaling factor. All values are then truncated, but if either `topLeft.x` or `extent.x` had a fractional part, `extent.x` is incremented by 1 (and, similarly, if either `topLeft.y` or `extent.y` had a fractional part, `extent.y` is incremented by 1).

If the active coordinate system is `kCoordinatesStandard`, `rectP` is not changed by this function.

Compatibility Implemented only if the High-Density Display Feature Set is present.

See Also `WinScaleRectangle()`, `WinUnscaleCoord()`, `WinUnscalePoint()`

WinValidateHandle Function

Purpose Return `true` if the specified handle references a valid window object.

Declared In `Window.h`

Prototype

```
Boolean WinValidateHandle (WinHandle winHandle)
```

Parameters → *winHandle*

The handle to be tested.

Returns Returns `true` if the specified handle references a non-`NULL` pointer to a window in the active window list, `false` if the handle references a window whose values are out of sync with the current system state.

Comments For debugging purposes only. Do not include this function in commercial applications.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `FrmValidatePtr()`, `FrmRemoveObject()`

WinWindowToDisplayPt Function

Purpose Convert a window-relative coordinate to a display-relative coordinate.

Declared In `Window.h`

Prototype

```
void WinWindowToDisplayPt (Coord *extentX,
    Coord *extentY)
```

Parameters

↔ *extentX*
Pointer to x coordinate to convert.

↔ *extentY*
Pointer to y coordinate to convert.

Returns Returns nothing.

Comments The coordinate passed is assumed to be relative to the draw window.

See Also `WinDisplayToWindowPt()`

60

Miscellaneous System Functions

This chapter describes miscellaneous system functions. The functions in this chapter are declared in the header files `Crc.h`, `DLServer.h`, `IntlMgr.h`, and `Localize.h`.

Crc16CalcBlock Function

Purpose Calculate the 16-bit CRC of a data block using the table lookup method.

Declared In `Crc.h`

Prototype

```
UInt16 Crc16CalcBlock (const void *bufP,
   UInt16 count, UInt16 crc)
```

Parameters

`bufP`
: Pointer to the data buffer.

`count`
: Number of bytes in the buffer.

`crc`
: Seed crc value.

Returns A 16-bit CRC for the data buffer.

DlkControl Function

Purpose Perform an operation at the behest of the desktop software. Among other things, this function is used to return values to the conduit during the handling of `sysAppLaunchCmdHandleSyncCallApp`.

Declared In `DLServer.h`

Prototype

```
Err DlkControl (DlkCtlEnum op, void *param1P,
    void *param2P)
```

Parameters

→ *op*
: Desktop Link control code. Use `dlkCtlSendCallAppReply` when sending a result back to the conduit while handling a `sysAppLaunchCmdHandleSyncCallApp` launch code.

↔ *param1P*
: Pointer to the first parameter (operation-specific). For `dlkCtlSendCallAppReply`, this parameter should point to a `DlkCallAppReplyParamType` structure.

↔ *param2P*
: Pointer to the second parameter (operation-specific). For `dlkCtlSendCallAppReply`, this parameter should be set to `NULL`.

Returns `errNone` if no error, or an error code if there was a problem during the call to `DlkControl`. In either case, place the value returned from `DlkControl` into the `replyErr` field of the `SysAppLaunchCmdHandleSyncCallAppType` structure when handling `sysAppLaunchCmdHandleSyncCallApp`.

Comments This function is needed to return data back to a conduit during the handling of `sysAppLaunchCmdHandleSyncCallApp`. Set `param1P` to point to a `DlkCallAppReplyParamType` structure, as described below. See the Example on page 1309 for an illustration of how to handle `sysAppLaunchCmdHandleSyncCallApp`.

DlkCallAppReplyParamType

Prototype

```
typedef struct DlkCallAppReplyParamType {
   UInt16 pbSize;
   UInt32 dwResultCode;
   const void *resultP;
   UInt32 dwResultSize;
   void *dlRefP;
   UInt32 dwReserved1;
} DlkCallAppReplyParamType;
```

Fields

`pbSize`
: Size of this parameter block. Set it to `sizeof(DlkCallAppReplyParamType)`.

`dwResultCode`
: Result code to be returned to the remote caller.

`resultP`
: Pointer to result data.

`dwResultSize`
: Size of result data block (number of bytes).

`dlRefP`
: Desktop Link reference pointer from `SysAppLaunchCmdHandleSyncCallAppType`.

`dwReserved1`
: Reserved. Set to `NULL`.

Example

The `SysAppLaunchCmdHandleSyncCallAppType` structure that accompanies the `sysAppLaunchCmdHandleSyncCallApp` launch code contains all of the information passed into `SyncCallRemoteModule` on the desktop as well as the necessary fields to pass the result pack to the desktop. At the end of your `sysAppLaunchCmdHandleSyncCallApp` launch code handler, you'll need to send a `DlkCallAppReplyParamType` reply structure back to the device using `DlkControl`.

```
#include <DLServer.h>
...
case sysAppLaunchCmdHandleSyncCallApp:
{
   SysAppLaunchCmdHandleSyncCallAppType *theCommandPtr;
   DlkCallAppReplyParamType theReplyParams;
   CharPtr theReplyBuffer = "SUCCESS";
```

```
   // Cast the cmdPBP to a SysAppLaunchCmdHandleSyncCallAppType
   // pointer so that we can work with it.
   theCommandPtr = (SysAppLaunchCmdHandleSyncCallAppType*)cmdPBP;

   // Do whatever work is necessary here.  If you set the m_wActionCode
   // field in your CCallModuleParams class on the desktop, then you
   // can handle that code by looking at the action field of theCommandPtr
   // (i.e.) if (theCommandPtr->action == 1)

   // Create the reply to send back to the desktop

   // First clear out all the fields.  This is necessary so that the reserved
   // fields are set to NULL.
   MemSet( &theReplyParams, sizeof(DlkCallAppReplyParamType), 0 );

   // Set the size of the reply.  This is required.
   theReplyParams.pbSize = sizeof(DlkCallAppReplyParamType);

   // Set the result code.  Normally this will be set to zero unless you want
   // to send an error code back to the desktop.
   theReplyParams.dwResultCode = 0;

   // Fill in the reply buffer and buffer length
   theReplyParams.resultP = theReplyBuffer;
   theReplyParams.dwResultSize = StrLen(theReplyBuffer) + 1;

   // Fill in the DL reference pointer.  This is required.
   theReplyParams.dlRefP = theCommandPtr->dlRefP;

   // Set the handled field to true.  This is required to let the desktop
   // know that the sysAppLaunchCmdHandleSyncCallApp was handled.  If you
   // don't set this to true, the call to SyncCallRemoteModule will return
   // SYNCERR_UNKNOWN_REQUEST.
   theCommandPtr->handled = true;

   // Finally, set the replyErr field by passing the reply parameters to
   // DlkControl.  This is required for the DLServer to properly handle the
   // reply request.
   theCommandPtr->replyErr = DlkControl (dlkCtlSendCallAppReply,
     &theReplyParams, NULL);

   break;
}
```

Table 60.1 and Table 60.2 list some important mappings from the `CCallModuleParams` class on the desktop to the

`SysAppLaunchCmdHandleSyncCallAppType` and `DlkCallAppReplyParamType` structures on the handheld.

Table 60.1 CCallModuleParams to SysAppLaunchCmdHandleSyncCallAppType mapping

CCallModuleParams	SysAppLaunchCmdHandleSyncCall AppType
m_wActionCode	action
m_dwParamSize	dwParamSize
m_pParam	paramP

Table 60.2 CCallModuleParams to DlkCallAppReplyParamType mapping

CCallModuleParams	DlkCallAppReplyParamType
m_dwResultBufSize	dwResultSize
m_pResultBuf	resultP
m_dwResultCode	dwResultCode

DlkGetSyncInfo Function

Purpose Get the sync info managed by Desktop Link. This function is often used to obtain the user name on the handheld.

Declared In `DLServer.h`

Prototype

```
Err DlkGetSyncInfo (UInt32 *succSyncDateP,
   UInt32 *lastSyncDateP,
   DlkSyncStateType *syncStateP, Char *nameBufP,
   Char *logBufP, Int32 *logLenP)
```

Parameters

← *succSyncDateP*
Pointer to the location where the date of the last successful sync is stored. Supply `NULL` for this parameter if this date isn't needed.

← *lastSyncDateP*
: Pointer to the location where the date of the last sync, successful or otherwise, is stored. Supply `NULL` for this parameter if this date isn't needed.

← *syncStateP*
: Pointer to a `DlkSyncStateType` enum into which the state of the last sync is stored. Supply `NULL` for this parameter if the state information isn't needed. See the Comments, below, for a description of this enum.

← *nameBufP*
: Pointer to a string buffer into which the null-terminated handheld user name is stored. This string buffer must have been preallocated to be at least `dlkUserNameBufSize` bytes in length. Supply `NULL` for this parameter if the user name isn't needed.

← *logBufP*
: Pointer to a string buffer into which the sync log text, null-terminated, is stored. Supply `NULL` for this parameter if the log text isn't needed. If you supply a valid pointer for this parameter, you must specify the preallocated buffer length using the `logLenP` parameter; the returned log text will be truncated, if necessary, to fit within the buffer.

↔ *logLenP*
: Pointer to the log buffer size. If `logBufP` is not `NULL`, on entry you must set this value to the size of the `logBufP` buffer. When this function returns, this value indicates the actual length of the log text, not counting the null terminator.

Returns Returns `errNone` if no error, or `dlkErrMemory` if the Desktop Link preferences resource couldn't be locked.

Comments The state information returned through `syncStateP` has one of the values defined by the `DlkSyncStateType` enum:

Prototype

```
typedef enum DlkSyncStateType {
  dlkSyncStateNeverSynced = 0,
  dlkSyncStateInProgress,
  dlkSyncStateLostConnection,
  dlkSyncStateLocalCan,
  dlkSyncStateRemoteCan,
  dlkSyncStateLowMemoryOnTD,
```

```
  dlkSyncStateAborted,
  dlkSyncStateCompleted,
  dlkSyncStateIncompatibleProducts,
  dlkSyncStateNPOD
} DlkSyncStateType;
```

Constants

`dlkSyncStateNeverSynced`
: The handheld has never been synced.

`dlkSyncStateInProgress`
: A sync is currently in progress.

`dlkSyncStateLostConnection`
: The connection was lost during sync.

`dlkSyncStateLocalCan`
: Sync was cancelled by the user on the handheld.

`dlkSyncStateRemoteCan`
: Sync was cancelled by the user from the desktop.

`dlkSyncStateLowMemoryOnTD`
: Sync ended due to a low memory condition on the handheld.

`dlkSyncStateAborted`
: Sync was aborted for some other reason.

`dlkSyncStateCompleted`
: Sync completed normally.

`dlkSyncStateIncompatibleProducts`
: Sync ended because the desktop HotSync product is incompatible with this version of the handheld HotSync.

`dlkSyncStateNPOD`
: The sync could not take place because the handheld has a 4.0-style password but the desktop hasn't yet been updated to a compatible version.

Example

This function is most often used to obtain the handheld user name. The following code excerpt shows how to do this (for clarity, error-checking has been omitted):

```
MemHandle nameH;
char *nameP;

// Allocate a buffer for the user name
nameH = MemHandleNew(dlkUserNameBufSize);
```

```
nameP = MemHandleLock(nameH);

// Obtain the user's name
DlkGetSyncInfo(NULL, NULL, NULL, nameP, NULL, NULL);

// ... Do something with the user name here ...

// Now that we're done with the user name, free the buffer
MemPtrUnlock(nameP);
```

Compatibility The `dlkSyncStateIncompatibleProducts` enum value was added in Palm OS 3.0. The `dlkSyncStateNPOD` enum value was added in Palm OS 4.0.

IntlGetRoutineAddress Function

Purpose Return the address of an international manager or text manager function.

Declared In `IntlMgr.h`

Prototype

```
void *IntlGetRoutineAddress
    (IntlSelector inSelector)
```

Parameters → *inSelector*
One of the routine selectors defined in `IntlMgr.h`.

Returns Returns the address of the corresponding function. Returns `NULL` if an invalid routine selector is passed.

Comments Use this function for performance reasons. It returns the address of an international manager or text manager function. You can then use this address to call the function without having to go through the international manager's trap dispatch table. This function is mostly useful for optimizing the performance of text manager routines that are called in a tight loop.

You might also use this function to check for the presence of newer international manager and text manager functions. If the result is `NULL`, the function is not implemented on this device.

Compatibility Implemented only if International Feature Set is present.

See Also `IntlSetRoutineAddress()`, `SysGetTrapAddress()`

IntlSetRoutineAddress Function

Purpose Set the address of the function corresponding to an international manager or text manager function.

Declared In `IntlMgr.h`

Prototype

```
Err IntlSetRoutineAddress (IntlSelector iSelector,
    void *iProcPtr)
```

Parameters

→ *iSelector*
One of the routine selectors defined in `IntlMgr.h`.

→ *iProcPtr*
Pointer to a function that the routine identified by `iSelector` should point to.

Returns Returns `errNone` if no error, or `intlErrInvalidSelector` if `iSelector` does not refer to a valid international manager or text manager routine.

Comments This function is useful for patching an international or text manager function. Normally only a locale module would need to patch one of these functions.

WARNING! If your application patches an international manager function using this function, you *must* remove the patch before your application exits. Do *not* use this mechanism to permanently patch international manager functions as it may cause unpredictable results for the system and other applications.

Compatibility Implemented only if 4.0 New Feature Set is present. If 5.0 New Feature Set is present this function is unimplemented.

See Also `IntlGetRoutineAddress()`, `SysSetTrapAddress()`

LocGetNumberSeparators Function

Purpose Get localized number separators.

Declared In `Localize.h`

Prototype

```
void LocGetNumberSeparators
    (NumberFormatType numberFormat,
    Char *thousandSeparator,
    Char *decimalSeparator)
```

Parameters

→ *numberFormat*
The format to use (see NumberFormatType).

← *thousandSeparator*
The character used for the thousands separator.

← *decimalSeparator*
The character used for the decimal separator.

Returns Returns nothing.

Comments The format to use is stored in the system preferences. You can obtain it by passing `prefNumberFormat` to `PrefGetPreference()`.

Compatibility Implemented only if 2.0 New Feature Set is present.

See Also `StrLocalizeNumber()`, `StrDelocalizeNumber()`, "Localized Applications" in the *Palm OS Programmer's Companion*, vol. I

PceNativeCall Function

Purpose Call a native ARM or Windows NT function from code running in the PACE (68k) environment.

Declared In `PceNativeCall.h`

Prototype

```
UInt32 PceNativeCall(NativeFuncType *nativeFuncP,
    void *userDataP)
```

Parameters

→ *nativeFuncP*
On a handheld with an ARM processor, this is a pointer to the ARM function to be executed. On Palm OS Simulator, this is a pointer to the name of a DLL and the name of the entry point within that DLL that is to be executed, separated by a null character and terminated with a null character. See the Comments, below, for more details on the format of this argument.

↔ *userDataP*
Pointer to an application-specific block of data that is passed to the ARM function. This block has no specific alignment requirements; it needn't be aligned on a 16- or 32-bit boundary. Note that your ARM function may impose specific alignment requirements, however.

Returns The return value of the specified ARM function is returned by `PceNativeCall`. This value is placed in both the A0 and D0 registers in the emulated 68k CPU, allowing `PceNativeCall` to support both pointer and immediate return value conventions.

Comments Applications that employ `PceNativeCall` won't work on handhelds running a version of Palm OS prior to Palm OS Garnet. Before calling `PceNativeCall`, your application must verify the underlying processor type, since the calling convention is different on Palm OS Simulator. See "Calling ARM-Native Subroutines" on page 406 of *Palm OS Programmer's Companion*, vol. I for more information and an example.

`PceNativeCall` byte-swaps the parameter pointer and return value as appropriate for the Dragonball-to-ARM transition. This allows you to dereference `userDataP` directly from your ARM code. Because the operating system has no knowledge of the structure of the parameter block, however, it performs no byte-swapping within this block. Your ARM code must do this as necessary for your application (see "Handle 68K and ARM Technical Differences" on page 410 of the *Palm OS Programmer's Companion*, vol. I for more information).

On Palm OS Simulator, rather than passing a pointer to a block of ARM code in `nativeFuncP`, you instead pass a pointer to the name of a DLL and the name of the function within that DLL that is to be executed. These two names must be separated by a null character, and the entire sequence must be terminated by a null character. For example, to load the DLL found at `C:\TEST_DLL\Debug\Simple.dll` and call the function `TestNativeCall` within that DLL, you might pass a pointer to the following character string literal:

```
"C:\\TEST_DLL\\Debug\\Simple.dll\0TestNativeCall"
```

Note that if you don't supply an absolute path, Simulator looks for the DLL in (or relative to) the directory from which `PalmSim.exe`

is running. Thus, if the DLL is located in the same directory as `PalmSim.exe`, you can call the above function with:

```
"Simple.dll\0TestNativeCall"
```

On release ROMs, `PceNativeCall` fails silently if `nativeFuncP` is NULL. On debug ROMs, it generates an error. All other pointers are treated as valid code and followed. If `nativeFuncP` is invalid, the processor will try to execute the code anyway and will eventually generate an error.

NOTE: The call to your native function is guaranteed to be made from ARM mode.

For more information on how the ARM code should be structured and how to call back and forth between the PACE and ARM environments, see "Palm OS Garnet ARM Programming" on page 403 of the *Palm OS Programmer's Companion*, vol. I.

Compatibility Implemented only if 5.0 New Feature Set is present.

Part III
Communications

61

Connection Manager

The Connection Manager allows other applications to access, add, and delete connection profiles contained in the Connection panel.

This chapter provides reference material for the Connection Manager API declared in the header file `ConnectionMgr.h`:

- Connection Manager Constants
- Connection Manager Functions

For more information on the Connection Manager, see the chapter "Serial Communication" on page 89 in the *Palm OS Programmer's Companion*, vol. II, *Communications*.

Connection Manager Data Types

CncProfileID Typedef

Purpose The `CncProfileID` type uniquely identifies a connection profile within the Connection Manager profile database. You pass this ID as a parameter to several of the Connection Manager functions. You can obtain a connection's profile ID using `CncProfileGetIDFromName()` or `CncProfileGetIDFromIndex()`.

Prototype

```
typedef UInt32 CncProfileID;
```

Compatibility Defined only if Connection Manager Feature Set is present.

Connection Manager Constants

General Profile Parameters

The Connection Manager defines the following constants to represent individual parameters in the preinstalled connection profiles. Not all parameters work for all types of profiles.

Parameter Name	Parameter Type	Description
`kCncParamBaud`	`UInt32`	The baud rate to use for the connection.
`kCncParamBluetoothDeviceAddr`	buffer	The 48-bit address (`BD_ADDR`) of the device that the handheld is connected to through the Bluetooth port. This parameter is only valid if `kCncParamPort` specifies the Bluetooth port.
`kCncParamBluetoothDeviceName`	string	The name of the device that the handheld is connected to through the Bluetooth port. The device name uses the UTF-8 character encoding. This parameter is only valid if `kCncParamPort` specifies the Bluetooth port.
`kCncParamCountryIndex`	`UInt16`	The index into the list of strings returned by `kCncParamIntlModemCountryStringList` and `kCncParamIntlModemResetStringList` that provides the name of the country and the reset commands for this profile.

`kCncParamCountryIndex`	`UInt16`	The index into the list of strings returned by `kCncParamIntlModemCountryStringList` and `kCncParamIntlModemResetStringList` that provides the name of the country and the reset commands for this profile.

Parameter Name	Parameter Type	Description
`kCncParamIntlModemCountryStringList`	buffer	For modem profiles, a data buffer containing a list of possible countries. This list contains the countries in which the modem can operate and is shown in the Connection panel Details form for this profile. The selected country controls the dialing prefix and the modem reset string. The `kCncParamCountryIndex` parameter specifies the country selected from this list. The data buffer contains a packed block of null-terminated strings, each containing a country name. The first 16 bits of the data buffer (a `UInt16` value) tells how many strings are contained in the block. You can use `SysFormPointerArrayToStrings()` to convert the data buffer into an array of strings. Because this parameter value has a variable size, you must first call `CncProfileSettingGet()` with a `NULL` data pointer to obtain the correct size.
`kCncParamIntlModemResetStringList`	buffer	For modem profiles, a data buffer containing possible reset strings. The `kCncParamCountryIndex` parameter specifies which reset string from this list is to be used. The actual string used is itself stored in the `kCncParamResetString` parameter.

Parameter Name	Parameter Type	Description
		The data buffer contains a packed block of null-terminated strings, each containing a reset string. The first 16 bits of the data buffer (a `UInt16` value) tells how many strings are contained in the block. You can use `SysFormPointerArrayToStrings()` to convert the data buffer into an array of strings. Because this parameter value has a variable size, you must first call `CncProfileSettingGet()` with a `NULL` data pointer to obtain the correct size.
`kCncParamInvisible`	system flag	If `true`, the profile is hidden from the user. If `false`, the profile is visible. This parameter is not currently used.
`kCncParamLocked`	system flag	If `true`, the profile is locked so that the user cannot edit it. If `false`, the profile can be edited. This parameter can be set by profiles, such as phone profiles, created by third party utilities.
`kCncParamName`	string	The name of the profile.
`kCncParamNoDetails`	system flag	If `true`, the profile details should not be displayed. If `false`, they can be displayed. The profile details are the parameters that appear in the Details form of the Connection panel.

Parameter Name	Parameter Type	Description
`kCncParamNonEditable`	system flag	If `true`, the Connection panel's Edit form should be suppressed for this profile. If `false`, the Edit form can be displayed. This parameter differs from `kCncParamLocked` and `kCncParamReadOnly` in that it causes an alert to be displayed if the user taps the Edit button. The other parameters allow the Edit form to be displayed but do not allow changes to be made. Also, a non-editable profile can be deleted, but a read-only or locked profile cannot.
`kCncParamPort`	`UInt32`	The logical, physical, or virtual port identifier. See "Port Constants" on page 1598 in the "Serial Manager" chapter for more information.
`kCncParamReadOnly`	system flag	If `true`, the profile is read-only and cannot be edited. If `false`, the profile can be edited. This parameter is only intended to be used by profiles pre-installed in the Palm OS.
`kCncParamResetString`	string	For modem and phone profiles, the reset string. For modem profiles, this is one of the strings in `kCncParamIntlModemResetStringList`.
`kCncParamSerialPortFlags`	`UInt32`	For phone profiles, bit flags that correspond to various serial port hardware settings. See "Serial Settings Constants" on page 1601 for more information.

Parameter Name	Parameter Type	Description
`kCncParamSystemFlags`	`UInt32`	A bit flag representing all system flags. Currently, only bits 0 through 4 have a meaning. These correspond to the read-only bit, the invisible bit, the noneditable bit, the no details bit, and the locked bit, respectively.
`kCncParamTimeOut`	`UInt32`	The amount of time in milliseconds to wait for a response when CTS is unasserted and hardware flow control is on.
`kCncParamVersion`	`UInt8`	The version of the Connection Manager API under which this profile was created. The current version number is `kCncProfileVersion`.
`kCncParamVolume`	`UInt16`	For modem profiles, the modem volume.

Compatibility Defined only if Connection Manager Feature Set is present.

Phone Profile Parameters

Purpose The Connection Manager defines the following constants to represent individual parameters in the preinstalled connection profiles for phones. Also see "General Profile Parameters" on page 1324 and "GPRS Profile Parameters" on page 1331 for other parameter constants that may apply.

Declared In `ConnectionMgr.h`

Constants

```
#define kCncParamReceiveTimeOut CncDefineParamID(
   kCncParamOSRange , kCncParamUInt32, 14 )
```

A `UInt32` containing the number of milliseconds to wait for a response from the phone. This time-out value is used by telephony functions that don't need to access the network (for example, `TelNwkGetSelectedNetwork()`).

```
#define kCncParamTTCreator CncDefineParamID(
  kCncParamOSRange, kCncParamUInt32, 91 )
```

A `UInt32` containing the creator ID for the telephony task used by the phone driver.

```
#define kCncParamTTType CncDefineParamID(
  kCncParamOSRange, kCncParamUInt32, 90 )
```

A `UInt32` containing the database type for the telephony task used by the phone driver.

```
#define kCncParam_GSM_BearerServices
  CncDefineParamID(kCncParamOSRange,
  kCncParamUInt8, 211 )
```

```
#define kCncParam_GSM_ConnectionElement
  CncDefineParamID(kCncParamOSRange,
  kCncParamUInt8, 212 )
```

```
#define kCncParam_GSM_Speed
  CncDefineParamID(kCncParamOSRange,
  kCncParamUInt8, 210 )
```

```
#define kCncParam_PSDCreator CncDefineParamID(
  kCncParamOSRange, kCncParamUInt32, 93 )
```

A `UInt32`.containing the creator ID of the phone driver.

```
#define kCncParam_PSDName CncDefineParamID(
  kCncParamOSRange, kCncParamString, 92 )
```

A string containing the name of the phone driver.

```
#define kCncParam_PSDParameterBuffer
  CncDefineParamID(kCncParamOSRange,
  kCncParamBuffer, 100 )
```

Aa data buffer containing any necessary data that the phone driver needs to store. This parameter typically holds data that is set using the Details form of the Connection panel.

```
#define kCncParam_PSDType
  CncDefineParamID(kCncParamOSRange,
  kCncParamUInt32, 94 )
```

A `UInt32`.containing the database type for the phone driver.

GPRS Profile Parameters

Purpose The Connection Manager defines the following constants to represent individual parameters in the preinstalled connection profiles for GPRS networks. Also see "General Profile Parameters" on page 1324 and "Phone Profile Parameters" on page 1329 for other parameter constants that may apply.

Declared In `ConnectionMgr.h`

Constants

```
#define kCncParam_GPRS_APNName
  CncDefineParamID(kCncParamOSRange,
  kCncParamString, 200 )
```

A string containing the GPRS access point name (APN).

```
#define kCncParam_GPRS_DataCompression
  CncDefineParamID(kCncParamOSRange,
  kCncParamUInt8, 202 )
```

A boolean value, stored in a `UInt8`, indicating data compression.

```
#define kCncParam_GPRS_HeaderCompression
  CncDefineParamID(kCncParamOSRange,
  kCncParamUInt8, 201 )
```

A boolean value, stored in a `UInt8`, indicating header compression.

```
#define kCncParam_GPRS_QOS_Delay
  CncDefineParamID(kCncParamOSRange,
  kCncParamUInt8, 204 )
```

A `UInt8` containing the GPRS quality-of-service delay value.

```
#define kCncParam_GPRS_QOS_Mean
  CncDefineParamID(kCncParamOSRange,
  kCncParamUInt8, 207 )
```

A `UInt8` containing the GPRS quality-of-service mean throughput.

```
#define kCncParam_GPRS_QOS_Peak
  CncDefineParamID(kCncParamOSRange,
  kCncParamUInt8, 206 )
```

A `UInt8` containing the GPRS quality-of-service peak throughput.

```
#define kCncParam_GPRS_QOS_Precedence
  CncDefineParamID(kCncParamOSRange,
  kCncParamUInt8, 203 )
```

A `UInt8` containing the GPRS quality-of-service precedence value.

```
#define kCncParam_GPRS_QOS_Reliability
  CncDefineParamID(kCncParamOSRange,
  kCncParamUInt8, 205 )
```

A `UInt8` containing the GPRS quality-of-service reliability value.

Profile Parameter Sizes

The size constants specify the size of each of the predefined parameters described in "General Profile Parameters." The following table lists the parameter name, the size of its value, and the size constant that gives this size. These size constants are suitable for passing to `CncProfileSettingGet()` and `CncProfileSettingSet()`.

Profile Parameter Name	Size	Size Constant
`kCncParamBaud`	32	`kCncParamBaudSize`
`kCncParamBluetoothDeviceAddr`	8	`kCncParamBluetoothDeviceAddrSize`
`kCncParamBluetoothDeviceName`	249	`kCncParamBluetoothDeviceNameMaxSize`
`kCncParamCountryIndex`	16	`kCncParamCountryIndexSize`
`kCncParamDeviceKind`	16	`kCncParamDeviceKindSize`
`kCncParamDialingMode`	8	`kCncParamDialingModeSize`
`kCncParamFlowControl`	16	`kCncParamFlowControlSize`
`kCncParamInitString`	81	`kCncParamInitStringMaxSize`
	81	`kCncProfileUsualInitStringSize`

Profile Parameter Name	Size	Size Constant
kCncParamInvisible	8	kCncParamInvisibleSize
kCncParamLocked	8	kCncParamLockedSize
kCncParamName	22	kCncParamNameMaxSize
	22	kCncProfileNameSize
kCncParamNoDetails	8	kCncParamNoDetailsSize
kCncParamNonEditable	8	kCncParamNonEditableSize
kCncParamPort	32	kCncParamPortSize
kCncParam_PSDCreator	32	kCncParam_PSDCreatorSize
kCncParam_PSDName	32	kCncParam_PSDNameSize
kCncParam_PSDType	32	kCncParam_PSDTypeSize
kCncParamReadOnly	8	kCncParamReadOnlySize
kCncParamReceiveTimeOut	32	kCncParamReceiveTimeOutSize
kCncParamResetString	81	kCncParamResetStringMaxSize
	8	kCncProfileClassicResetStringSize
	81	kCncProfileUsualResetStringSize
kCncParamSystemFlags	32	kCncParamSystemFlagsSize
kCncParamTimeOut	32	kCncParamTimeOutSize
kCncParamTTCreator	32	kCncParamTTCreatorSize
kCncParamTTType	32	kCncParamTTTypeSize
kCncParamVersion	8	kCncParamVersionSize
kCncParamVolume	16	kCncParamVolumeSize
kCncParam_GPRS_APNName		kCncParam_GPRS_APNNameMaxSize (this is the maximum size that the APN name string may have).

Profile Parameter Name	Size	Size Constant
kCncParam_GPRS_DataCompression	8	kCncParam_GPRS_DataCompressionSize
kCncParam_GPRS_HeaderCompression	8	kCncParam_GPRS_HeaderCompressionSize
kCncParam_GPRS_QOS_Delay	8	kCncParam_GPRS_QOS_DelaySize
kCncParam_GPRS_QOS_Mean	8	kCncParam_GPRS_QOS_MeanSize
kCncParam_GPRS_QOS_Peak	8	kCncParam_GPRS_QOS_PeakSize
kCncParam_GPRS_QOS_Precedence	8	kCncParam_GPRS_QOS_PrecedenceSize
kCncParam_GPRS_QOS_Reliability	8	kCncParam_GPRS_QOS_ReliabilitySize
kCncParam_GSM_BearerServices	8	kCncParam_GSM_BearerServicesSize
kCncParam_GSM_ConnectionElement	8	kCncParam_GSM_ConnectionElementSize
kCncParam_GSM_Speed	8	kCncParam_GSM_SpeedSize

Compatibility Defined only if Connection Manager Feature Set is present.

Profile Parameter Types

Purpose The parameter type constants specify the type of value stored for a parameter in a connection profile. If you define a new parameter using `CncDefineParamID()`, you must use one of these constants to specify the type of data the parameter stores. The macro `CncGetParamType()` can return this information for any parameter in the profile.

Declared In `ConnectionMgr.h`

Constants

`#define kCncParamBuffer`
`CncDefineParameterType(kCncParamVariableLen, 2)`
A generic block of data.

`#define kCncParamString`
`CncDefineParameterType(kCncParamVariableLen, 1)`
A string parameter.

`#define kCncParamSystemFlag`
`CncDefineParameterType(kCncParamFixedLen, 0)`
A system flag parameter. The Connection Manager can store up to 32 system flags. Flags are stored in a single bit and returned as a `UInt8` value. The entire system flags word can be returned if you pass `kCncParamSystemFlags` to `CncProfileSettingGet()`.

`#define kCncParamUInt16`
`CncDefineParameterType(kCncParamFixedLen, 2)`
A `UInt16` parameter.

`#define kCncParamUInt32`
`CncDefineParameterType(kCncParamFixedLen, 3)`
A `UInt32` parameter.

`#define kCncParamUInt8`
`CncDefineParameterType(kCncParamFixedLen, 1)`
A `UInt8` parameter.

Compatibility Defined only if Connection Manager Feature Set is present.

Device Kinds

Purpose The device kind constants specify the type of connection being made. You specify the type of connection by defining a value for the `kCncParamDeviceKind` parameter using `CncProfileSettingSet()`.

Declared In `ConnectionMgr.h`

Constants

`#define kCncDeviceKindGPRS 4`
The connection is to a GPRS network.

`#define kCncDeviceKindGSM 5`
The connection is to GSM phone.

`#define kCncDeviceKindLocalNetwork 3`
The connection is to a LAN.

`#define kCncDeviceKindModem 1`
The connection is to a modem.

`#define kCncDeviceKindPhone 2`
The connection is to a phone.

`#define kCncDeviceKindSerial 0`
The connection is through the serial port.

Compatibility Defined only if Connection Manager Feature Set is present.

Connection Manager Functions

CncAddProfile Function

Purpose Adds a profile to the Connection Manager.

Declared In `ConnectionMgr.h`

Prototype

```
Err CncAddProfile (Char *name, UInt32 port,
   UInt32 baud, UInt16 volume, UInt16 handShake,
   const Char *initString,
   const Char *resetString,
   Boolean isModem, Boolean isPulse)
```

Parameters

↔ *name*
Pointer to the profile name to be added. If the name is already taken in the Connection panel then a duplication number is appended to it. The name added is returned here.

→ *port*
The port identification used by the profile. See "Specifying the Port" on page 100 of the *Palm OS Programmer's Companion*, vol. II, *Communications* for more information.

→ *baud*
The baud rate used by the profile.

→ *volume*
The volume setting for the device (for Modem only).

→ *handShake*
Flow control setting (hardware handshaking). 0 specifies automatic (on at speeds > 2400 baud), 1 specifies always on, and 2 specifies always off.

→ *initString*
Pointer to the initialization string used by a modem (for Modem only).

→ *resetString*
Pointer to the reset string used by a modem (for Modem only).

→ *isModem*
`true` if Modem, `false` if Direct.

→ *isPulse*
`true` if Pulse dial, `false` if TouchTone.

Returns

`errNone`
No error.

`cncErrAddProfileFailed`
The add operation failed.

`cncErrProfileListFull`
The add operation failed because the profile list is full.

`cncErrConDBNotFound`
The connection database is missing.

Comments

All profiles within the Connection Manager must have a unique name. The Connection Manager tries to append a duplication number to the end of the name if you specify a name that is already taken.

There is a maximum limit to the number of profiles that can be maintained by the Connection Manager. If the limit is passed, an error is returned and that profile will not be added.

Profiles that do not need certain fields may pass `NULL` in the place of a value.

Compatibility

Implemented only if New Serial Manager Feature Set is present.

If Connection Manager Feature Set is present, use `CncProfileCreate()` instead of using this function. `CncAddProfile` is still supported for backward compatibility. In

Palm OS 4.0 and higher, the maximum number of profiles that can be defined has greatly increased.

Example

```
AddMyProfile()
{
   Char *myConNameP;
   Err err;

   myConNameP = MemPtrNew(cncProfileNameSize);

   StrCopy(myConNameP, "Foobar");

   err = CncAddProfile(myConNameP, 'u328', 57600, 0, 0,
      "AT&FX4", 0,  true, false);

   MemPtrFree(myConNameP);
}
```

CncDefineParamID Macro

Purpose Creates and returns a parameter ID.

Declared In `ConnectionMgr.h`

Prototype

```
#define CncDefineParamID (parameterRange,
    parameterType,
    parameterID)
```

Parameters

→ *parameterRange*
: Use `kCncParamThirdPartiesRange` to specify that this parameter is not defined by the OS.

→ *parameterType*
: The type of value stored for the parameter. See "Profile Parameter Types" for a list of possible values.

→ *parameterID*
: A unique value between 0 to 1023. The value must be unique within the profile for which you are defining the parameter.

  If you are using a `parameterType` of `kCncParamSystemFlag`, specify a value from 0 to 31 to identify which of the system flag bit is to be set.

Returns Returns the parameter ID as a `UInt16` value.

Comments You use this macro only if you are defining your own connection profile and have a parameter that you need to define within that

profile. The parameter ID immediately precedes its parameter value in the Connection Manager profile database. Because of how the database is formatted, the parameter ID must tell the Connection Manager how to interpret the next series of bytes. For this reason, the high order bits of the parameter ID include information about the type of value and whether the value is defined by the system or a third party.

Compatibility Parameter IDs of this format are only used if 4.0 New Feature Set is present.

See Also `CncProfileSettingSet()`, `CncProfileSettingGet()`

CncDeleteProfile Function

Purpose Removes a profile from the Connection Manager.

Declared In `ConnectionMgr.h`

Prototype

```
Err CncDeleteProfile (const Char *name)
```

Parameters

→ *name*
Pointer to the name of the profile to be deleted.

Returns

`errNone`
No error.

`cncErrProfileReadOnly`
The profile could not be deleted because it is read only.

`cncErrProfileNotFound`
The profile could not be found.

`cncErrConDBNotFound`
The connection database is missing.

Comments The profiles that come preinstalled on the unit are read only and cannot be deleted.

Compatibility Implemented only if New Serial Manager Feature Set is present.

If Connection Manager Feature Set is present, use `CncProfileDelete()` instead of using this function.

CncGetParamType Macro

Purpose Returns the parameter type portion of the parameter ID.

Declared In `ConnectionMgr.h`

Prototype

```
#define CncGetParamType (parameterID)
```

Parameters

→ *parameterID*
A `UInt16` that contains the parameter ID.

Returns Returns a `UInt16` value where bits 11 through 14 contain one of the values in "Profile Parameter Types" and the other bits are clear.

Compatibility Parameter IDs of this format are only used if 4.0 New Feature Set is present.

See Also `CncProfileSettingSet()`, `CncProfileSettingGet()`

CncGetProfileInfo Function

Purpose Returns the settings for a profile.

Declared In `ConnectionMgr.h`

Prototype

```
Err CncGetProfileInfo (Char *name, UInt32 *port,
   UInt32 *baud, UInt16 *volume,
   UInt16 *handShake,
   Char *initString, Char *resetString,
   Boolean *isModem, Boolean *isPulse)
```

Parameters

→ *name*
Pointer to the name of the profile to be returned. Passing in `NULL` causes this function to return the settings for the profile currently selected in the Connection panel.

← *port*
Pointer to the port identifier that the profile uses.

← *baud*
Pointer to the baud rate that has been set for this profile.

← *volume*
Pointer to the volume of the device (applies only to modems).

← *handShake*
Pointer to the flow control setting (hardware handshaking). 0 indicates automatic (on at speeds > 2400 baud), 1 indicates always on, and 2 indicates always off.

← *initString*
Pointer to the initialization string for the device (applies only to modems).

← *resetString*
Pointer to the reset string for the device (applies only to modems).

← *isModem*
Pointer to a Boolean value: `true` for Modem, `false` for Direct.

← *isPulse*
Pointer to a Boolean value: `true` for Pulse dial, `false` for TouchTone.

Returns

`errNone`
No error.

`cncErrGetProfileFailed`
The get profile operation failed. The profile may or may not be there.

`cncErrProfileNotFound`
The profile could not be found

`cncErrConDBNotFound`
The connection database is missing.

Comments

One or more of the parameters may be set to `NULL` if that information is not desired.

Compatibility

Implemented only if New Serial Manager Feature Set is present.

If Connection Manager Feature Set is present, use `CncProfileSettingGet()` with one of the "General Profile Parameters" instead of using this function.

Example

```
{
     UInt32 portID, baud;
     UInt16 openPort;
     // get port id
     err = CncGetProfileInfo("Direct Serial", &portID, &baud,
        0, 0, 0, 0, 0, 0);
```

```
     if(!err)
     { // open the port
        SrmOpen(portID, baud, &openPort);
     }
}
```

CncGetProfileList Function

Purpose Returns a list of available profiles that are available through the Connection Manager.

Declared In `ConnectionMgr.h`

Prototype

```
Err CncGetProfileList (Char ***nameListPPP,
   UInt16 *countP)
```

Parameters

← *nameListPPP*
Pointer to a pointer to a list of profile names.

← *countP*
Pointer to the number of profile names.

Returns

`errNone`
No error.

`cncErrGetProfileListFailed`
The profile list could not be found.

`cncErrConDBNotFound`
The connection database is missing.

Comments Allocation of the list is handled by the Connection Manager; deallocation is the responsibility of the calling application. Appended to the end of the list will be "-Current-", which represents the profile currently selected in the Connection panel.

Compatibility Implemented only if New Serial Manager Feature Set is present.

Example

```
//Declared globally
Char **globalProfileList;
ListType *listP;
UInt16 globalProfileCount;

void SetConnectionList()
{
   //Get the list from the Connection Manager
   err = CncGetProfileList(&globalProfileList,
      &globalProfileCount);
```

```
   //Set the UI list
   LstSetListChoices(listP, globalProfileList,
      globalProfileCount);
}

void StopApplication()
{
   UInt16 i;

   //Deallocate the connection list
   For(i = 0; i < globalProfileCount;  i++)
      MemPtrFree(globalProfileList[ i ]);
   MemPtrFree(globalProfileList);
}
```

CncGetSystemFlagBitnum Macro

Purpose Returns the number uniquely identifying a system flag parameter.

Declared In `ConnectionMgr.h`

Prototype

```
#define CncGetSystemFlagBitnum (parameterID)
```

Parameters

→ *parameterID*
The `UInt16` containing the parameter ID.

Returns Returns the ID of the system flag parameter, which is a value from 0 to 31.

Compatibility Parameter IDs of this format are only used if Connection Manager Feature Set is present.

See Also `CncProfileSettingSet()`, `CncProfileSettingGet()`

CncGetTrueParamID Macro

Purpose Returns the portion of the parameter ID that uniquely identifies the parameter.

Declared In `ConnectionMgr.h`

Prototype

```
#define CncGetTrueParamID (parameterID)
```

Parameters

→ *parameterID*
A `UInt16` containing the parameter ID.

Returns Returns a `UInt16` containing just the parameter ID. The high-order bits, which specify the parameter type and address space, are clear.

Compatibility Parameter IDs of this format are only used if Connection Manager Feature Set is present.

See Also `CncProfileSettingSet()`, `CncProfileSettingGet()`

CncIsFixedLengthParamType Macro

Purpose Specifies whether the parameter value is fixed length or variable length.

Declared In `ConnectionMgr.h`

Prototype

```
#define CncIsFixedLengthParamType (parameterID)
```

Parameters → *parameterID*
A `UInt16` containing the parameter ID.

Returns Returns `true` if the parameter is a fixed length parameter type such as `UInt32`, or `false` if it is a variable length type.

Compatibility Parameter IDs of this format are only used if Connection Manager Feature Set is present.

See Also `CncProfileSettingSet()`, `CncProfileSettingGet()`

CncIsSystemFlags Macro

Purpose Returns whether the parameter value is a system flag.

Declared In `ConnectionMgr.h`

Prototype

```
#define CncIsSystemFlags (parameterID)
```

Parameters → *parameterID*
The `UInt16` containing the parameter ID.

Returns Returns `true` if the parameter type is a system flag. Returns `false` otherwise.

Compatibility Parameter IDs of this format are only used if Connection Manager Feature Set is present.

See Also `CncProfileSettingSet()`, `CncProfileSettingGet()`

CncIsSystemRange Macro

Purpose Specifies whether the parameter is in the system range or in the third-party range.

Declared In `ConnectionMgr.h`

Prototype

```
#define CncIsSystemRange (parameterID)
```

Parameters → *parameterID*
A `UInt16` containing the parameter ID.

Returns Returns `true` if the parameter ID is defined by Palm OS, or `false` if it is defined by a third party.

Compatibility Parameter IDs of this format are only used if Connection Manager Feature Set is present.

See Also `CncProfileSettingSet()`, `CncProfileSettingGet()`

CncIsThirdPartiesRange Macro

Purpose Specifies whether the parameter is defined by a third party.

Declared In `ConnectionMgr.h`

Prototype

```
#define CncIsThirdPartiesRange (parameterID)
```

Parameters → *parameterID*
A `UInt16` containing the parameter ID.

Returns Returns `true` if the parameter is a third-party parameter, or `false` if it is a system parameter.

Compatibility Parameter IDs of this format are only used if Connection Manager Feature Set is present.

See Also `CncProfileSettingSet()`, `CncProfileSettingGet()`

CncIsVariableLengthParamType Macro

Purpose Returns whether the parameter value is a variable-length type.

Declared In `ConnectionMgr.h`

Prototype

```
#define CncIsVariableLengthParamType
    (parameterID)
```

Parameters → *parameterID*
A `UInt16` containing the parameter ID.

Returns Returns `true` if the parameter is a variable-length string or a buffer or `false` if it holds a fixed-length type such as an integer.

Compatibility Parameter IDs of this format are only used if Connection Manager Feature Set is present.

See Also `CncProfileSettingSet()`, `CncProfileSettingGet()`

CncProfileCloseDB Function

Purpose Closes the Connection Manager profile database.

Declared In `ConnectionMgr.h`

Prototype

```
Err CncProfileCloseDB (void)
```

Parameters `None.`

Returns

`errNone`
No error.

`kCncErrDBAccessFailed`
The database could not be closed or this is a reference counting error.

Comments Use `CncProfileOpenDB()` and `CncProfileCloseDB` as an optimization if you make several Connection Manager calls in succession. All Connection Manager calls open the profile database when they begin and close the database when they are finished. The Connection Manager maintains a reference count that tells it whether the database is open. If you call `CncProfileOpenDB` before making another Connection Manager call, the next call does not open or close the database. This saves your application the overhead of opening and closing the database each time a call is made.

Compatibility Implemented only if Connection Manager Feature Set is present.

CncProfileCount Function

Purpose Returns the number of connection profiles currently defined in the Connection Manager profile database.

Declared In `ConnectionMgr.h`

Prototype

```
Err CncProfileCount (UInt16 *profilesCountP)
```

Parameters

← *profilesCountP*
The number of profiles.

Returns

`errNone`
No error.

`kCncErrDBAccessFailed`
The profile database could not be opened.

Compatibility Implemented only if Connection Manager Feature Set is present.

See Also `CncGetProfileList()`

CncProfileCreate Function

Purpose Adds a profile record to the Connection Manager profile database.

Declared In `ConnectionMgr.h`

Prototype

```
Err CncProfileCreate (CncProfileID *profileIdP)
```

Parameters

← *profileIdP*
Upon return, the unique ID of the new profile.

Returns

`errNone`
No error.

`kCncErrDBAccessFailed`
The profile database could not be opened.

`a Data Manager error`
The new record could not be created.

Comments This function creates a new empty record in the Connection Manager profile database. To populate the profile, use `CncProfileSettingSet()` to set parameter values, including the

profile name. Use `CncDefineParamID()` if you need to store information unique to your profile.

Compatibility Implemented only if Connection Manager Feature Set is present.

See Also `CncAddProfile()`, `CncProfileDelete()`

CncProfileDelete Function

Purpose Deletes a profile.

Declared In `ConnectionMgr.h`

Prototype

```
Err CncProfileDelete (CncProfileID profileId)
```

Parameters

`profileId`
The ID of the profile to delete.

Returns

`errNone`
No error.

`kCncErrDBAccessFailed`
The profile database could not be opened, or the record could not be deleted.

`kCncErrProfileParamNotFound`
The database does not contain a profile with the specified ID.

Comments The profiles that come preinstalled on the unit are read only and cannot be deleted.

Compatibility Implemented only if Connection Manager Feature Set is present.

See Also `CncDeleteProfile()`, `CncProfileCreate()`

CncProfileGetCurrent Function

Purpose Returns the ID of the currently selected profile in the Connection panel.

Declared In `ConnectionMgr.h`

Prototype

```
Err CncProfileGetCurrent
    (CncProfileID *profileIdP)
```

Parameters

← *profileIdP*
The ID of the current profile.

Returns

`errNone`
No error.

`kCncErrDBAccessFailed`
The profile database could not be opened.

Compatibility Implemented only if Connection Manager Feature Set is present.

See Also `CncProfileGetIDFromIndex()`, `CncProfileGetIDFromName()`, `CncProfileGetIndex()`, `CncProfileSetCurrent()`

CncProfileGetIDFromIndex Function

Purpose Returns the profile ID given its index into the Connection Manager profile database.

Declared In `ConnectionMgr.h`

Prototype

```
Err CncProfileGetIDFromIndex (UInt16 index,
   CncProfileID *profileIdP)
```

Parameters

→ *index*
The index of the Connection Manager profile.

← *profileIdP*
The ID of the Connection Manager profile.

Returns

`errNone`
No error.

`kCncErrDBAccessFailed`
The profile database could not be opened.

`kCncErrProfileParamNotFound`
No profile at that index.

Compatibility Implemented only if Connection Manager Feature Set is present.

See Also `CncProfileGetIDFromName()`, `CncProfileGetCurrent()`, `CncProfileGetIndex()`

CncProfileGetIDFromName Function

Purpose Returns the profile ID given its name.

Declared In `ConnectionMgr.h`

Prototype

```
Err CncProfileGetIDFromName
    (const Char *profileNameP,
    CncProfileID *profileIdP)
```

Parameters

→ *profileNameP*
The name of the profile. The name is displayed in a pop-up list in the Connection panel. If you pass the string "-Current-", this function returns the ID of the current profile.

← *profileIdP*
The profile ID.

Returns

`errNone`
No error.

`kCncErrDBAccessFailed`
The profile database could not be opened.

`kCncErrProfileParamNotFound`
No profile with the specified name.

Compatibility Implemented only if Connection Manager Feature Set is present.

See Also `CncProfileGetCurrent()`, `CncProfileGetIDFromIndex()`, `CncProfileGetIndex()`

CncProfileGetIndex Function

Purpose Returns the index of the profile given its ID.

Declared In `ConnectionMgr.h`

Prototype

```
Err CncProfileGetIndex (CncProfileID profileId,
    UInt16 *indexP)
```

Parameters

→ *profileId*
The profile ID.

← *indexP*
The index of the profile's record in the Connection Manager profile database.

Returns

`errNone`
No error.

`kCncErrDBAccessFailed`
The profile database could not be opened.

`kCncErrProfileParamNotFound`
No profile with the specified ID.

Compatibility Implemented only if Connection Manager Feature Set is present.

See Also `CncProfileGetIDFromIndex()`

CncProfileOpenDB Function

Purpose Opens the Connection Manager profile database.

Declared In `ConnectionMgr.h`

Prototype

```
Err CncProfileOpenDB (void)
```

Parameters None

Returns

`errNone`
No error.

`kCncErrDBAccessFailed`
The profile database could not be opened.

Comments Use `CncProfileOpenDB` and `CncProfileCloseDB()` as an optimization if you make several Connection Manager calls in succession. All Connection Manager calls open the profile database when they begin and close the database when they are finished. The Connection Manager maintains a reference count that tells it whether the database is open. If you call `CncProfileOpenDB` before making another Connection Manager call, the next call does not open or close the database. This saves your application the overhead of opening and closing the database each time a call is made.

The Connection Manager profile database is created if it does not exist.

Compatibility Implemented only if Connection Manager Feature Set is present.

CncProfileSetCurrent Function

Purpose Sets the current profile.

Declared In `ConnectionMgr.h`

Prototype

```
Err CncProfileSetCurrent (CncProfileID profileId)
```

Parameters

→ *profileId*
The ID of the profile to be made current.

Returns

`errNone`
No error.

`kCncErrDBAccessFailed`
The profile database could not be opened.

Comments The current profile is the profile that is used for the next network connection attempt.

Compatibility Implemented only if Connection Manager Feature Set is present.

See Also `CncProfileGetCurrent()`

CncProfileSettingGet Function

Purpose Obtains a value stored in one of the Connection Manager profiles.

Declared In `ConnectionMgr.h`

Prototype

```
Err CncProfileSettingGet (CncProfileID profileId,
   UInt16 paramId, void *paramBufferP,
   UInt16 *ioParamSizeP)
```

Parameters

→ *profileId*
The ID of the profile from which to obtain a parameter value.

→ *paramId*
The ID of the parameter to obtain. See "General Profile Parameters" for a list of the parameters used in the profiles that come preinstalled on the device.

← *paramBufferP*
A pointer to a buffer into which to write the parameter value. If the parameter stores a variable-sized value, you can determine the necessary size by passing `NULL` for `paramBufferP`. Upon return, `paramSize` contains the required size.

↔ *ioParamSizeP*
On input, a pointer to the size of the buffer into which to write the parameter. On output, points to the number of bytes written to `paramBufferP`.

Returns

`errNone`
No error.

`kCncErrDBAccessFailed`
The profile database could not be opened.

`kCncErrProfileGetParamFailed`
The Connection Manager failed to obtain the value of the parameter.

`kCncErrProfileBadSystemFlagBitnum`
An attempt was made to obtain the value of a system flag that is undefined.

`kCncErrProfileBadParamSize`
The `paramBufferP` buffer is too small. `ioParamSizeP` contains the correct size for the buffer.

`kCncErrProfileParamNotFound`
The specified parameter is not defined in the profile.

Compatibility Implemented only if Connection Manager Feature Set is present.

See Also `CncProfileSettingSet()`, `CncGetProfileInfo()`

CncProfileSettingSet Function

Purpose Sets a parameter value in the specified profile.

Declared In `ConnectionMgr.h`

Prototype

```
Err CncProfileSettingSet
    (CncProfileID iProfileId,
    UInt16 paramId, const void *paramBufferP,
    UInt16 paramSize)
```

Parameters

→ *iProfileId*
The ID of the profile.

→ *paramId*
The ID of the parameter. See "General Profile Parameters" for a list of the parameters defined in the preinstalled connection profiles.

→ *paramBufferP*
A pointer to the value to set for this parameter.

→ *paramSize*
The size of the buffer passed in `paramBufferP`. See "Profile Parameter Sizes."

Returns

`errNone`
No error.

`kCncErrDBAccessFailed`
The profile database could not be opened.

`kCncErrProfileParamNotFound`
No profile with the specified ID.

`kCncErrProfileSetParamFailed`
The Connection Manager failed to set the value of the parameter.

`kCncErrProfileBadParamSize`
The `paramBufferP` buffer is too small. Most likely, the passed in size does not allow space for a string parameter's null terminator.

`kCncErrProfileParamNameHasChange`
An attempt was made to set the profile name to a name that is already used. The Connection Manager appends a duplication number to the end of the name and returns this error. You should use `CncProfileSettingGet()` to find out the new name.

Compatibility

Implemented only if Connection Manager Feature Set is present.

See Also

`CncDefineParamID()`, `CncProfileSettingGet()`

62

Exchange Manager

This chapter describes the Exchange Manager API declared in the header file `ExgMgr.h` and the Exchange Local Library API declared in the header file `ExgLocalLib.h`. It discusses the following topics:

- Exchange Manager Data Structures
- Exchange Manager Constants
- Exchange Manager Functions
- Application-Defined Functions

For more information on the Exchange Manager, see the chapter "Object Exchange" on page 1 of *Palm OS Programmer's Companion*, vol. II, *Communications*.

Exchange Manager Data Structures

ExgAskResultType Enum

Purpose

The `ExgAskResultType` enum defines possible values for the `result` field of the `sysAppLaunchCmdExgAskUser` launch code parameter block.

Prototype

```
typedef enum {
  exgAskDialog,
  exgAskOk,
  exgAskCancel
} ExgAskResultType;
```

Constants

`exgAskDialog`
: The Exchange Manager should display a dialog that prompts the user to confirm the receipt of data. See `ExgDoDialog()`.

`exgAskOk`
: Accept the data.

`exgAskCancel`
: Reject the data.

ExgGoToType Struct

Purpose The `ExgGoToType` structure defines the `goToParams` field of the `ExgSocketType` structure. Applications that want to be launched after the data is received place their creator IDs in the `goToCreator` field and define the `goToParams` field. The values in this structure are copied to the `sysAppLaunchCmdGoTo` launch code's parameter block.

Prototype

```
typedef struct {
  UInt16 dbCardNo;
  LocalID dbID;
  UInt16 recordNum;
  UInt32 uniqueID;
  UInt32 matchCustom;
} ExgGoToType;
```

Fields

`dbCardNo`
: The card number of the database that contains the added record.

`dbID`
: The local ID of the database that contains the added record.

`recordNum`
: The index of the record that was added.

`uniqueID`
: The unique ID of the record that was added. This field is not used.

`matchCustom`
: Application-specific information.

ExgLocalSocketInfoType Struct

Purpose The `ExgLocalSocketInfoType` structure identifies information specific to the Local Exchange Library. The `socketRef` field of the `ExgSocketType` structure is set to this structure when you send and receive data using the Local Exchange Library. The Local

Exchange Library creates this structure if it does not already exist. You only need to create it if you want to supply non-default values for the `noAsk` or `previewInfoP` fields.

Prototype

```
typedef struct {
  Boolean freeOnDisconnect;
  Boolean noAsk;
  ExgPreviewInfoType *previewInfoP;
  FileHand tempFileH;
  Err err;
  ExgLocalOpType op;
} ExgLocalSocketInfoType;
```

Fields

`freeOnDisconnect`
: Whether the structure is freed when the `ExgDisconnect()` call is made. The default is `true`. In general, code that allocates a structure should be responsible for freeing that structure. Therefore, if you have allocated `ExgLocalSocketInfoType`, you should set this field to `false` and explicitly free the structure when you are finished with it.

`noAsk`
: Set to `true` to disable the display of the exchange dialog. If you want to, for example, create a vCalendar object and send it to the Datebook application in response to a user command, you probably want to set `noAsk` to `true` so that the user does not have to confirm the receipt of the data they just requested you to send.

`previewInfoP`
: A pointer to an `ExgPreviewInfoType` structure, used to display a preview of the data. If you wanted to simply use another application to help display data, you would create and initialize this structure.

`tempFileH`
: A temporary buffer that the Local Exchange Library uses. Do not set this field directly; the Local Exchange Library should set it.

`err`
: The error code returned from the Local Exchange Library. Do not set this field directly; the Local Exchange Library should set it.

`op`
: The operation in progress. Do not set this field directly. The Local Exchange Library sets this field to one of the following:

`exgLocalOpNone`
: No operation in progress.

`exgLocalOpPut`
: A send is in progress.

`exgLocalOpAccept`
: A receive is in progress.

`exgLocalOpGet`
: A get is in progress.

`exgLocalOpGetSender`
: The library is receiving information from the sending application during a get operation.

ExgPreviewInfoType Struct

Purpose The `ExgPreviewInfoType` structure provides information to the `ExgNotifyPreview()` function. The `ExgNotifyPreview` function uses this information to have the application display a preview of the data to be received in the exchange dialog.

Prototype

```
typedef struct {
  UInt16 version;
  ExgSocketType *socketP;
  UInt16 op;
  Char *string;
  UInt32 size;
  RectangleType bounds;
  UInt16 types;
  Err error;
} ExgPreviewInfoType;
```

Fields

`version`
: Set this field to 0 to specify version 0 of this structure.

`socketP`
: A pointer to the socket structure (see `ExgSocketType`). The `libraryRef` field must identify the exchange library from

which preview data should be received, and the `target`, `type`, or `name` field should be defined as well.

`op`

One of the following constants:

`exgPreviewDialog`
: Display a modal dialog containing the preview. This constant is only used in situations where one application launches another to display data.

`exgPreviewDraw`
: The preview is a graphic.

`exgPreviewLongString`
: The preview is a long string.

`exgPreviewQuery`
: Ask the application which preview operations it supports. The answer is returned in the `types` field. If the application does not support any preview modes, the `error` field contains `exgErrNotSupported`.

`exgPreviewShortString`
: The preview is a short string.

`string`

A buffer into which the application places the string preview if `exgPreviewLongString` or `exgPreviewShortString` is specified.

`size`

The allocated size of the `string` field.

`bounds`

The bounds of the rectangle in which the application draws the graphic if `exgPreviewDraw` is specified.

`types`

Upon return from an `exgPreviewQuery` operation, a bit field identifying the types of previews the application supports.

`error`

The error code returned from the application. If `errNone`, the preview operation was successful.

Applications can define and use their own constants for the preview operation. Operations specific to an application are numbered starting at `exgPreviewFirstUser` and should be no greater than `exgPreviewLastUser`.

Compatibility This structure is only defined if 4.0 New Feature Set is present.

ExgSocketType Struct

Purpose The `ExgSocketType` structure defines an Exchange Manager socket, which is passed to most Exchange Manager functions. The `ExgSocketPtr` type points to a `ExgSocketType` structure.

Prototype

```
typedef struct ExgSocketType {
  UInt16 libraryRef;
  UInt32 socketRef;
  UInt32 target;
  UInt32 count;
  UInt32 length;
  UInt32 time;
  UInt32 appData;
  UInt32 goToCreator;
  ExgGoToType goToParams;
  UInt16 localMode:1;
  UInt16 packetMode:1;
  UInt16 noGoTo:1;
  UInt16 noStatus:1;
  UInt16 preview:1;
  UInt16 reserved:11;
  Char *description;
  Char *type;
  Char *name;
} ExgSocketType;
typedef ExgSocketType *ExgSocketPtr;
```

Note that when data is received, some of the fields in this structure may not have values. When you are sending data, it is recommended that you provide values for all of these fields, but you should not rely on receiving values for the fields marked optional.

Fields

`libraryRef`
The exchange library in use. When an application or library receives a socket, this field is already assigned.

When sending data, applications may identify the exchange library they want to connect with by providing a URL in the `name` field. If so, they should use 0 for the `libraryRef` field. The Exchange Manager then determines which library corresponds to the URL and assigns the `libraryRef` field. See the Comments in the `ExgPut()` function description for more information.

`socketRef`
The connection identifier. This value is supplied by the exchange library when a connection is established. It contains any necessary library-specific data.

`target`
The creator ID of the application that should receive the message.

`count`
The number of objects in this connection, usually 1 (optional).

`length`
The total byte count for all objects being sent (optional).

`time`
The last modified time of object (optional).

`appData`
Application-specific information (optional).

`goToCreator`
The creator ID of the application to launch using the `sysAppLaunchCmdGoTo` launch code after the item is received if `noGoTo` is 0. The value is assigned by the application that receives the object. See the Comments section in `ExgDisconnect()` for more information.

`goToParams`
If `goToCreator` is specified, then this field contains data that is copied into the launch code's parameter block. See `ExgGoToType`.

`localMode`
: Set to 1 to exchange with local device only. A `localMode` of 1 is equivalent to specifying a URL with the `exgLocalPrefix`. Set to 0 to enable an exchange with a remote machine. The default is 0.

`packetMode`
: Set to 1 to use connectionless packet mode (Ultra). The default is 0. Ultra mode is not currently supported.

`noGoTo`
: Set to 1 to disable launching the application with `sysAppLaunchCmdGoTo`. The default is 0.

`noStatus`
: If `true`, the exchange library should not display a progress dialog. If `false`, the library can display a progress dialog. The default is `false`.

  The Exchange Manager sets and clears this bit at various times while data is received. Applications may also want to set this bit if they use the Local Exchange Library and want to prevent the progress dialog from being displayed during a send.

`preview`
: If `true`, a preview is in progress. The `ExgNotifyPreview()` function sets this bit while the preview takes place and clears it when the preview is finished. Exchange libraries should not discard any data while a preview is in progress because the full data must be sent later if the receiving user accepts it.

`reserved`
: Reserved system flags.

`description`
: A pointer to the text description of the object (optional).

`type`
: A pointer to the MIME type of the object (optional).

`name`
: The name of the object being sent. This can be a URL whose scheme identifies the exchange library to connect with.

  If the name has a colon, it is treated as a URL.

Compatibility The `noGoTo` and `noStatus` flags are only defined if 3.5 New Feature Set is present, and the `noStatus` flag has no effect unless 4.0 New Feature Set is present. The `preview` flag is only defined if 4.0 New Feature Set is present.

Exchange Manager Constants

Registry ID Constants

The registry ID constants are used in the Exchange Manager registry. Exchange libraries register for the URL prefixes they handle, and applications register for the types of data they receive. The registry ID constants specify which type of data is being registered for.

Constant	Value	Description
`exgRegCreatorID`	0xfffb	Register for a creator ID. The `target` field of the `ExgSocketType` contains the creator ID of the application that should receive the data. Typically, the application with the matching creator ID receives the data, but it is possible for one application to register for another's creator ID and receive data in its place.
`exgRegSchemeID`	0xfffc	Register for a URL scheme. Typically, only exchange libraries register for URL schemes. Applications can register for URL schemes, but they only receive the URL when `ExgRequest()` is called. If the `name` field of the `ExgSocketType` contains a colon (:), the portion of the URL before the colon is the URL scheme. The default library registered for URLs with that scheme will handle the message.

Constant	Value	Description
`exgRegExtensionID`	0xfffd	Register for a filename extension. If the `name` field of the `ExgSocketType` contains a period (.), the portion of the name after the last period is the filename extension. The application registered to handle files of that extension will handle the message.
`exgRegTypeID`	0xfffe	Register for a MIME type. If the `type` field of the `ExgSocketType` contains a value, the application registered to receive that MIME type handles the message.

Compatibility The `exgRegCreatorID` and `exgRegSchemeID` constants are only defined if 4.0 New Feature Set is present.

Predefined URL Schemes

The Exchange Manager provides these predefined URL schemes, for which exchange libraries can register.

Constant	Value	Description
`exgBeamScheme`	"_beam"	The URL scheme for Beam commands. By default, the IR Library handles this scheme.
`exgSendScheme`	"_send"	The URL scheme for Send commands. The purpose of the Send command is to provide a choice of transport mechanisms to the user; therefore, any exchange library that sends data should register for this scheme.
`exgLocalScheme`	"_local"	The URL scheme for the Local Exchange Library.

Compatibility These constants are only defined if 4.0 New Feature Set is present.

Predefined URL Prefixes

The Exchange Manager provides the following prefixes, which can be used to construct URLs appropriate for the `name` field of the `ExgSocketType` structure. When sending data, applications provide a URL to identify the exchange library that should transport the data.

Constant	Value	Description
`exgBeamPrefix`	(`exgBeamScheme` ":")	The URL to beam data.
`exgSendPrefix`	("?" `exgSendScheme` ":")	A URL for the general Send command. Because this URL begins with a question mark (?), the Exchange Manager displays a dialog with a list of exchange libraries registered for the `exgSendScheme`. The user then chooses the desired exchange library.
`exgSendBeamPrefix`	("?" `exgSendScheme` ";" `exgBeamScheme` ":")	A URL for the general Send command. The Exchange Manager displays a dialog with a list of exchange libraries registered for either the `exgSendScheme` or the `exgBeamScheme`.
`exgLocalPrefix`	(`exgLocalScheme` ":")	The URL for using the Local Exchange Library.

Compatibility These constants are only defined if 4.0 New Feature Set is present.

Exchange Manager Functions

ExgAccept Function

Purpose Accepts a connection from a remote device.

Declared In `ExgMgr.h`

Prototype

```
Err ExgAccept (ExgSocketType *socketP)
```

Parameters

→ *`socketP`*
A pointer to the socket structure (see `ExgSocketType`).

Returns Returns one of the following error codes:

`errNone`	Success
`exgErrBadLibrary`	Couldn't find default exchange library
`exgErrNotSupported`	A preview is in progress, and the exchange library identified by `libraryRef` doesn't support preview mode

Other error codes depend on the exchange library.

Displays a fatal error message if `socketP` does not have a `libraryRef` specified.

Comments Applications call this function when launched with the `sysAppLaunchCmdExgReceiveData` or `sysAppLaunchCmdExgPreview` launch code. The launch code contains `socketP` in its parameter block. Applications should pass this socket to `ExgAccept` to accept the connection, then call `ExgReceive()` one or more times to receive the data, and then call `ExgDisconnect()` to disconnect.

NOTE: Don't create the socket on the receiving side of an exchange. The socket is passed to you in the command parameter block of the `sysAppLaunchCmdExgReceiveData` or `sysAppLaunchCmdExgPreview` launch code.

Compatibility Implemented only if 3.0 New Feature Set is present. Preview mode is supported only if 4.0 New Feature Set is present.

See Also `ExgConnect()`, `ExgPut()`, `ExgGet()`

ExgConnect Function

Purpose Establishes a connection with a remote socket.

Declared In `ExgMgr.h`

Prototype

```
Err ExgConnect (ExgSocketType *socketP)
```

Parameters

→ *`socketP`*
A pointer to the socket structure (see `ExgSocketType`). Specify either a value for the `libraryRef` field or a URL in the `name` field. `libraryRef` should be 0 if the `name` field contains a URL.

Returns Returns one of the following error codes:

`errNone`	Success
`exgErrBadLibrary`	Couldn't find exchange library
`exgErrNotSupported`	The library doesn't support the operation specified in `socketP`
`exgErrUserCancel`	The user cancelled the connection operation
`exgMemError`	There isn't enough free memory to respond to the request
`exgErrNotEnoughPower`	The battery does not have enough power to perform the operation

Other error codes depend on the exchange library.

Comments Applications can call this function to initiate a connection for sending multiple objects or for performing two-way communications. Some exchange libraries support sending multiple objects but do not support this call. See "Sending Multiple Objects" on page 16 of *Palm OS Programmer's Companion*, vol. II, *Communications* for more information.

Before calling this function, the application must initialize the `socketP` parameter. The socket should identify the exchange library to connect with by providing either a library reference number in the `libraryRef` field or a URL in the `name` field. The default exchange library registered for that type of URL handles the connection.

To provide users with a choice of transport mechanisms, specify a URL that begins with a question mark (?). The Exchange Manager displays a dialog with a list of all exchange libraries that respond to URLs of the specified type. If only one exchange library is registered for this URL scheme, no dialog is displayed.

For example, many applications on Palm OS® 4.0 or higher support a Send command. This command generates a URL with the prefix `exgSendPrefix` (see Predefined URL Prefixes). The Exchange Manager displays a dialog containing a list of libraries registered for that URL scheme. The user selects an exchange library, and that library's `ExgLibConnect` function is called.

If the library is not specified by either URL or library reference number (in the `libraryRef` field), the Exchange Manager by default uses the IR Library; however, if the `localMode` flag is set, the Local Exchange Library is used instead.

In addition to specifying the library, you can set the `count` field in `socketP` before making this call to indicate the number of objects that are going to be sent. Use a `count` of 0 if the number of objects isn't known in advance.

If no error is returned from `ExgConnect`, applications can follow this call either by sending multiple objects or requesting data from the remote device or both. To send an object, call `ExgPut()` at the beginning of each object and call `ExgSend()` one or more times per object to send the data. To request data from the remote device, use `ExgGet()` (and then use `ExgReceive()` to receive the requested data). You can use these calls in combination with each other to support two-way communications. After all of the objects have been sent and received, call `ExgDisconnect()` to disconnect.

IMPORTANT: Not all exchange libraries support the sending of multiple objects or using `ExgGet` to request data.

Compatibility Implemented only if 3.0 New Feature Set is present. `ExgConnect` was for system use only until the release of Palm OS 4.0. Multiple object sending and identifying exchange libraries by URL are supported only if 4.0 New Feature Set is present. On earlier releases, this function is an alias for the `ExgPut` function.

See Also `ExgPut()`, `ExgAccept()`, `ExgGet()`

ExgControl Function

Purpose Requests that an exchange library perform an operation.

Declared In `ExgMgr.h`

Prototype

```
Err ExgControl (ExgSocketType *socketP,
   UInt16 op,
   void *valueP, UInt16 *valueLenP)
```

Parameters

→ *socketP*
A pointer to the socket structure (see `ExgSocketType`). Specify either a value for the `libraryRef` field or a URL in the `name` field. `libraryRef` should be 0 if the `name` field contains a URL.

→ *op*
A constant identifying the operation that the exchange library should perform. See the Comments section for more information.

↔ *valueP*
Upon entry, a parameter that the exchange library requires to perform the operation, if any. Most operations do not require an input parameter. Upon return, contains the result of the operation.

↔ *valueLenP*
The size of the `valueP` buffer. The size is updated upon return to show the actual length of the content returned.

Returns Returns one of the following error codes:

`errNone`	Success

`exgErrBadLibrary`	Couldn't find the requested exchange library
`exgErrNotSupported`	The exchange library does not support the requested operation

Other error codes depend on the exchange library.

Comments

The Exchange Manager uses this function to request information from the exchange library. Applications may also call this function.

The Exchange Manager defines and uses a set of operation constants that it might send to any exchange library. These constants begin with the prefix `exgLibCtlGet`. The type of the variable pointed to by `valueP` depends on the type of operation to be performed. Table 62.1 lists and describes the predefined Exchange Manager operations.

Table 62.1 ExgControl operations for all exchange libraries

Operation exgLibCtlGet...	value Data Type	Description
`Preview`	Boolean. Output only.	Returns `true` if the exchange library supports preview mode or `false` if not. If the exchange library does not respond to this operation, it is assumed to support preview mode.
`Title`	String buffer of size `exgTitleBufferSize` bytes. Output only.	Returns the name of the exchange library as it should appear in the Send dialog. All exchange libraries must respond to this operation.
`Version`	`UInt16`. Output only.	Returns the version of the exchange library API that this library implements. The constant `exgLibAPIVersion` defines the current version number. If the exchange library does not respond to this operation, the library supports the version of the Exchange Library API defined in Palm OS 4.0.

An exchange library may also define its own operations. For example, the IR Library supports operations to enable or disable beaming, to set the baud rates, or to use the serial port (see "IR Control Constants" on page 1429). The SMS Library supports operations that allow you to set the SMS preferences for sending messages or to manipulate multipart messages (see "SMS Control Constants" on page 2182). Operations specific to an exchange library are numbered starting at `exgLibCtlSpecificOp`.

The `socketP` passed to this function must identify an exchange library either using the `libraryRef` field or using a URL in the `name` field. The Comments section in `ExgConnect()` describes how an application should identify the exchange library.

Compatibility Implemented only if 4.0 New Feature Set is present.

ExgDBRead Function

Purpose Converts a Palm OS database from its internal format and writes it to storage RAM.

Declared In `ExgMgr.h`

Prototype

```
Err ExgDBRead (ExgDBReadProcPtr readProcP,
    ExgDBDeleteProcPtr deleteProcP,
    void *userDataP, LocalID *dbIDP,
    UInt16 cardNo, Boolean *needResetP,
    Boolean keepDates)
```

Parameters

→ *readProcP*
A pointer to a function that reads in the database and passes it to `ExgDBRead`. See `ExgDBReadProcPtr()` for details.

→ *deleteProcP*
A pointer to a function that is called if a database with an identical name already exists on the device. See `ExgDBDeleteProcPtr()` for details.

→ *userDataP*
A pointer to any data you want to pass to either the `readProcP` or `deleteProcP` functions. Often, this parameter is used to pass the `ExgSocketType` that is required by many Exchange Manager functions.

← *dbIDP*
The ID of the database that `ExgDBRead` created on the local device.

→ *cardNo*
The number of the card on which the database is to be stored.

← *needResetP*
Set to `true` by `ExgDBRead` if the `dmHdrAttrResetAfterInstall` attribute bit is set in the received database.

→ *keepDates*
Specify `true` to retain the creation, modification, and last backup dates as set in the received database header. Specify `false` to reset these dates to the current date.

Returns

Returns `errNone` if successful; otherwise, returns one of the data manager error codes (`dmErr...`) or a callback-specific error code. (If the `readProcP` function returns an error, it is also returned by `ExgDBRead`.)

Comments

This function converts data received from an exchange library or from any other transport mechanism into a Palm OS database and stores that database in the storage heap. It is not required that you use this function in conjunction with Exchange Manager calls. That is, it's possible to use this function to perform other operations, such as converting a database created on the desktop computer to a Palm OS formatted database in the storage heap.

The primary use of this function, however, is to receive a database that has been beamed onto the device. In this case, call `ExgDBRead` in response to the launch code `sysAppLaunchCmdExgReceiveData` after calling `ExgAccept()` to accept the connection. Place the call to `ExgReceive()` in the read callback function you passed as the `readProcP` parameter. Pass the `ExgSocketType` structure returned from `ExgAccept` in the `userDataP` parameter so that you have access to it in the read callback function.

The read callback function performs the actual reading of data. `ExgDBRead` calls the read callback function multiple times. Each time, the `sizeP` parameter contains the number of bytes `ExgDBRead` expects the data returned in `dataP` to contain. It's important for the read callback function to set the number of bytes

(in `sizeP`) that it actually placed in `dataP` if it's not the same as what `ExgDBRead` expected. `ExgDBRead` stops calling the read callback function after 0 is returned in `sizeP`.

The callback function you pass in `deleteProcP` handles the case where the database being read already exists on the device. It is called only in that circumstance. The callback function may want to close the database if it is open, change the existing database's name, or delete the existing database to allow an overwrite. See `ExgDBDeleteProcPtr()` for more information.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `ExgDBWrite()`

ExgDBWrite Function

Purpose Converts a given Palm OS database from its internal format on the local device and writes it using a function you supply.

Declared In `ExgMgr.h`

Prototype

```
Err ExgDBWrite (ExgDBWriteProcPtr writeProcP,
    void *userDataP, const char *nameP,
    LocalID dbID, UInt16 cardNo)
```

Parameters

→ *writeProcP*
A pointer to a function that writes out the database identified by `dbID`. See `ExgDBWriteProcPtr()` for details.

→ *userDataP*
A pointer to any data you want to pass to the `writeProcP` function. Often, this parameter is used to pass the `ExgSocketType` that is required by many Exchange Manager functions.

→ *nameP*
A pointer to the name of the database that you want `ExgDBWrite` to write. This database is passed to `writeProcP`.

→ *dbID*
The ID of the database that you want `ExgDBWrite` to pass to `writeProcP`. If you don't supply an ID, then `nameP` is used to search for the database by name.

→ *cardNo*
The number of the card on which to look for the database identified by `nameP`.

Returns Returns `errNone` if successful; otherwise, returns one of the data manager error codes (`dmErr...`) or a callback-specific error code. (If the `writeProcP` function returns an error, it is also returned by `ExgDBWrite`.)

Comments This function converts a Palm OS formatted database on the storage heap into a stream of bytes that can be sent over the Internet or over any other transport mechanism. It is not required that you use this function in conjunction with Exchange Manager calls.

The primary use of this function, however, is to write a database that is going to be beamed onto another device. In this case, call `ExgDBWrite` after establishing the connection with `ExgPut()`. Place the call to `ExgSend()` in the write callback function you passed as the `writeProcP` parameter. Pass the `ExgSocketType` structure returned from `ExgSend` in the `userDataP` parameter so that you have access to it in the write callback function.

The write callback function performs the actual writing of data. `ExgDBWrite` calls the write callback function multiple times. Each time, the `sizeP` parameter contains the number of bytes of `dataP` that are to be written. If the write callback function didn't handle it all, it's important that it set in `sizeP` the number of bytes that it did handle successfully. `ExgDBWrite` stops calling the write callback function after 0 is returned in `sizeP`.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `ExgDBRead()`

ExgDisconnect Function

Purpose Terminates an Exchange Manager transfer and disconnects.

Declared In `ExgMgr.h`

Prototype

```
Err ExgDisconnect (ExgSocketType *socketP,
    Err error)
```

Parameters

→ *socketP*
A pointer to the socket structure (see `ExgSocketType`) identifying the connection to terminate.

→ *error*
Any error that occurred. This parameter tells the exchange library why the connection is being broken. Normally the error code from `ExgSend()` or `ExgReceive()` is passed in here.

Returns Returns one of the following error codes:

`errNone`	Success
`exgErrBadLibrary`	Couldn't find default exchange library
`exgMemError`	Couldn't read data to send
`exgErrUserCancel`	User cancelled transfer

Other error codes depend on the exchange library.

May display a fatal error message if `socketP` doesn't contain a `libraryRef` value.

Comments Applications must call this function when finished sending data or receiving data. It terminates the connection made with `ExgConnect()`, `ExgAccept()`, `ExgPut()`, or `ExgGet()`.

In the `error` parameter, pass any error that occurs during the application loop, including errors returned from other Exchange Manager functions. This ensures that the connection is shut down knowing that it failed rather than succeeded.

It's especially important to check the result code from this function, since this will tell you if the transfer was successful. An `errNone` return value means that the item was delivered to the destination successfully. It does not mean that the user on the other end actually kept the data.

`ExgDisconnect` is used after sending and receiving. When receiving, the application can insert its creator ID into the `goToCreator` field in the socket structure and add other goto information in the `goToParams` field. After the application returns from the `sysAppLaunchCmdExgReceiveData` launch code, the exchange library may call `ExgNotifyGoto()`, which launches the `goToCreator` application with the standard launch code `sysAppLaunchCmdGoTo`.

IMPORTANT: Placing your creator ID in the `goToCreator` field is no longer a guarantee that you receive this launch code starting in Palm OS 4.0 because Palm OS 4.0 supports the sending of multiple objects at once. Thus, another application might overwrite the `goToCreator` field after your application has disconnected, making that application the recipient of the launch code.

Note that some exchange libraries wait to establish a connection until `ExgDisconnect` is called. The IR Library, for example, buffers the data that it receives and then waits until `ExgDisconnect` to actually send this data unless `ExgConnect` is called to establish a multi-object send connection.

Compatibility Implemented only if 3.0 New Feature Set is present.

Prior to Palm OS release 4.0, the Exchange Manager always launched the `goToCreator` application, if one was provided, upon return from this function. If 4.0 New Feature Set is present, the Exchange Manager does not launch the `goToCreator` application. Exchange libraries that want the previous behavior must explicitly call `ExgNotifyGoto()`.

See Also `ExgReceive()`, `ExgSend()`

ExgDoDialog Function

Purpose Displays a dialog that allows users to accept or reject the receipt of data.

Declared In `ExgMgr.h`

Prototype

```
Boolean ExgDoDialog (ExgSocketType *socketP,
   ExgDialogInfoType *infoP, Err *errP)
```

Parameters

→ *socketP*

A pointer to the socket structure (see `ExgSocketType`) identifying the connection. A value must be provided for the `libraryRef` field.

Applications can obtain the socket structure from the `sysAppLaunchCmdExgAskUser` launch code parameter block.

↔ *infoP*

A pointer to an `ExgDialogInfoType` structure (see the Comments section below).

← *errP*

`errNone` if no error, or the error code if an error occurred. Currently, no errors are returned.

Returns Returns `true` if the user clicks the OK button on the dialog, or `false` otherwise.

Comments This function displays the exchange dialog, which prompts the user to accept or reject incoming data.

By default, the Exchange Manager calls this function if the receiving application doesn't handle the `sysAppLaunchCmdExgAskUser` launch code or if it returns `exgAskDialog` from the launch code handler. When the Exchange Manager calls `ExgDoDialog`, the dialog displays a message similar to "Do you want to accept 'John Doe' into Address Book?" and allows the user to accept or reject the data. If the user clicks OK, the data should be received as an unfiled record.

The Exchange Manager attempts to display a preview of the data in the exchange dialog to provide users with enough information to determine if they want to accept or reject the data. To display the preview data, it calls `ExgNotifyPreview()`. Applications wishing to support preview mode should respond to the launch code

sysAppLaunchCmdExgPreview. See the ExgNotifyPreview function's description for more information.

Applications may also want to allow users to select a category in which to accept the incoming data. To do so, handle sysAppLaunchCmdExgAskUser to call ExgDoDialog directly and pass it a pointer to an ExgDialogInfoType structure. The ExgDialogInfoType structure is defined as follows:

Prototype

```
typedef struct {
  UInt16 version;
  DmOpenRef db;
  UInt16 categoryIndex;
} ExgDialogInfoType;
```

Parameters

→ *version*
Set this field to 0 to specify version 0 of this structure.

→ *db*
A pointer to an open database that defines the categories the dialog should display.

← *categoryIndex*
The index of the category in which the user wants to file the incoming data.

If db is valid, the function extracts the category information from the specified database and displays it in a pop-up list. Upon return, the categoryIndex field contains the index of the category the user selected, or dmUnfiledCategory if the user did not select a category.

If the call to ExgDoDialog is successful, your application is responsible for retaining the value returned in categoryIndex and using it to file the incoming data as a record in that category. One way to do this is to store the categoryIndex in the socket's appData field (see ExgSocketType) and then extract it from the socket in your response to the launch code sysAppLaunchCmdExgReceiveData. For example:

```
if (cmd == sysAppLaunchCmdExgReceiveData) {
  UInt16 category =
    (ExgSocketPtr)cmdPBP->appData;
  /* other declarations */

/* Receive the data, and create a new record
```

```
  using the received data. indexNew is the
  index of this record. */
  if (category !- dmUnfiledCategory) {
    UInt16 attr;
    Err err;
    err = DmRecordInfo(dbP, indexNew, &attr,
      NULL, NULL);

    // Set the category to the one the user
    // specified, and mark the record dirty.
    if ((attr & dmRecAttrCategoryMask) !=
      category) {
      attr &= ~dmRecAttrCategoryMask;
      attr |= category | dmRecAttrDirty;
      err = DmSetRecordInfo(dbP, indexNew,
        &attr, NULL);
    }
  }
}
```

Some of the Palm OS built-in applications (Address Book, Memo, and ToDo) use this method of setting the category on data received through beaming. Refer to the example code for these applications provided in the SDK for a more complete example of how to use `ExgDoDialog`.

When you explicitly call `ExgDoDialog`, you must set the `result` field of the `sysAppLaunchCmdExgAskUser` launch code's parameter block to either `exgAskOk` (upon success) or `exgAskCancel` (upon failure) to prevent the system from displaying the dialog a second time.

Compatibility

Implemented only if 3.5 New Feature Set is present.

Preview mode display in the exchange dialog is implemented only if 4.0 New Feature Set is present.

ExgGet Function

Purpose Establishes a connection and requests an object from a remote device.

Declared In `ExgMgr.h`

Prototype

```
Err ExgGet (ExgSocketType *socketP)
```

Parameters → *`socketP`*

A pointer to the socket structure (see `ExgSocketType`). Specify either a value for the `libraryRef` field or a URL in the `name` field. `libraryRef` should be 0 if the `name` field contains a URL. The `target`, `type`, or `name` fields should identify the data being requested.

Returns Returns one of the following error codes:

`errNone`	Success
`exgErrBadLibrary`	Couldn't find default exchange library
`exgErrUserCancel`	The user cancelled the operation
`exgMemError`	There is not enough free memory to perform the operation

Other error codes depend on the exchange library.

Comments Applications use this function to request data (initiate a send) from a remote device. Not all exchange libraries support this operation; for example, neither the IR Library nor Bluetooth support it.

Before calling this function, the application must initialize the `socketP` parameter. The socket should identify the exchange library to connect with by providing either a library reference number in the `libraryRef` field or a URL in the `name` field. The default exchange library registered for the URL's scheme handles the connection. The socket should also specify what data it is requesting by providing values for at least one of the `target`, `name`, and `type` fields. Specifying the data in the `name` field is the most common method.

To provide users with a choice of transport mechanisms, the application can provide a URL that begins with a question mark (?). The Exchange Manager displays a dialog with a list of all exchange libraries that respond to URLs of the specified type. If only one

exchange library is registered for this URL scheme, no dialog is displayed.

If the library is not specified by either URL or library reference number, the Exchange Manager by default uses the IR Library; however, if the `localMode` flag is set, the Local Exchange Library is used instead.

Applications can use `ExgGet` to initiate a send from the Local Exchange Library. For more information, see "Sending and Receiving Locally" on page 32 of the *Palm OS Programmer's Companion*, vol. II, *Communications*.

If no error is returned, applications should follow this call with one or more calls to `ExgReceive()`, to receive the data, or `ExgDisconnect()`, to disconnect.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `ExgPut()`, `ExgConnect()`

ExgGetDefaultApplication Function

Purpose Retrieves the default application for the specified type of data or the default exchange library for URLs with the specified scheme.

Declared In `ExgMgr.h`

Prototype

```
Err ExgGetDefaultApplication (UInt32 *creatorIDP,
    UInt16 id, const Char *dataTypeP)
```

Parameters

← *creatorIDP*
: A pointer to the creator ID of the default application or default exchange library.

→ *id*
: The registry ID constant identifying the type of data in `dataTypeP`. See Registry ID Constants.

→ *dataTypeP*
: A pointer to a string that contains the type of data for which to retrieve the default application or library. If `dataTypeP` is a file extension, do not include the period (.). If it is a URL, do not include the colon (:).

Returns Returns `errNone` if a match was found or `exgErrNoKnownTarget` if there is no default application or library for this type of data.

Comments You might use this function to see which application on this device will receive a particular type of data or to see which library on this device handles URLs of a particular scheme.

For example, to find out which application receives TXT files on this device, do the following:

```
UInt32 creatorID;
Err error;
error = ExgGetDefaultApplication(&creatorID,
  exgRegExtensionID, "TXT");
if (!error) {
    //creatorID contains default application.
```

To find out which exchange library handles URLs that use the beam prefix, do the following:

```
UInt32 creatorID;
Err error;
error = ExgGetDefaultApplication(&creatorID,
  exgRegSchemeID, exgBeamScheme);
if (!error) {
    //creatorID contains default library.
```

It's possible to have several applications registered to receive the same type of data, but none of them is the default. When the Exchange Manager receives an object of that type, it selects an application to receive the data, and it selects that same application every time. The selected application effectively becomes the default for the data type even though it is not explicitly set as the default. If this is the case, the `ExgGetDefaultApplication` function returns the creator ID of this de-facto default application.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `ExgGetRegisteredApplications()`, `ExgGetRegisteredTypes()`, `ExgRegisterDatatype()`, `ExgSetDefaultApplication()`

ExgGetRegisteredApplications Function

Purpose Retrieves a list of all applications registered to receive data of a specified type.

Declared In `ExgMgr.h`

Prototype

```
Err ExgGetRegisteredApplications
    (UInt32 **creatorIDsP, UInt32 *numAppsP,
    Char **namesP, Char **descriptionsP,
    UInt16 id, const Char *dataTypeP)
```

Parameters

← *creatorIDsP*
An array of the creator IDs of the applications registered to receive objects of this type. Pass `NULL` for this parameter if you only want to know how many applications are registered for this type.

← *numAppsP*
The number of applications registered to receive objects of this type. This is the number of elements in the `creatorIDsP` array, the `namesP` array, and the `descriptionsP` array.

← *namesP*
A packed list of strings, suitable for passing to `SysFormPointerArrayToStrings()`, containing the names of the applications or libraries. Each string is no more than `exgMaxTitleLen` characters. Pass `NULL` for this parameter if you don't want to retrieve it.

← *descriptionsP*
A packed list of strings, suitable for passing to `SysFormPointerArrayToStrings`, containing the descriptions of the applications or libraries. Descriptions are specified when the applications or libraries register for data. Each string is no more than `exgMaxDescriptionLength` characters. Pass `NULL` for this parameter if you don't want to retrieve it.

→ *id*
The registry ID constant identifying the type of data in `dataTypeP`. See Registry ID Constants.

→ *dataTypesP*
A pointer to a tab-delimited, null-terminated string listing the items to register. (Use `\t` for the tab character.) Each item

in the string must be no more than `exgMaxTypeLength` characters. There can be no more than 16 types total.

Returns Returns `errNone` upon success or `exgMemError` if the function cannot allocate space for the creator IDs, names, or descriptions.

IMPORTANT: This function allocates enough space for the `creatorIDsP`, `namesP`, and `descriptionsP` arrays as long as you do not pass `NULL` for the parameters. You are still responsible for freeing these arrays.

Comments You might use this function to see which applications on this device can receive a particular type of data or to see which libraries on this device handle URLs of a particular scheme. You can also use it to built a list of choices from which the user can select a default application or default exchange library for a particular data type or URL scheme. For example, iMessenger uses this function to build a list of mailto handlers so that the user can choose one of them to be the default.

The Exchange Manager itself uses `ExgGetRegisteredApplications` to find exchange libraries when it is given a URL that begins with a question mark (?). It displays the returned list to the user in the Send With dialog.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `ExgGetDefaultApplication()`, `ExgGetRegisteredTypes()`, `ExgRegisterDatatype()`, `ExgSetDefaultApplication()`

ExgGetRegisteredTypes Function

Purpose Retrieve a list of all data types for which a registration exists.

Declared In `ExgMgr.h`

Prototype

```
Err ExgGetRegisteredTypes (Char **dataTypesP,
    UInt32 *sizeP, UInt16 id)
```

Parameters ← *dataTypesP*

A packed list of strings, suitable for passing to `SysFormPointerArrayToStrings()`, containing a sorted

list of data types for which a registration exists. Each string is no more than `exgMaxTypeLength` characters.

← *sizeP*
The number of elements in the `dataTypesP` array.

→ *id*
The type of data to search for. For example, you can search for all registered creator IDs, all registered MIME types, and so on.

Returns Returns `errNone` upon success or `exgMemError` if the function cannot allocate space for the data types array.

IMPORTANT: This function allocates enough space for the `dataTypesP` array as long as you do not pass `NULL` for the parameter. You are still responsible for freeing this array.

Comments This function could be used to create an application that allows users to choose the default application for each data type.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `ExgGetDefaultApplication()`, `ExgGetRegisteredApplications()`, `ExgRegisterDatatype()`, `ExgSetDefaultApplication()`

ExgGetTargetApplication Function

Purpose Retrieves the application that should receive a specific message. This function does not search for libraries.

Declared In `ExgMgr.h`

Prototype

```
Err ExgGetTargetApplication
    (ExgSocketType *socketP, Boolean unwrap,
    UInt32 *creatorIDP, Char *descriptionP,
    UInt32 descriptionSize)
```

Parameters → *socketP*
A pointer to the socket structure (see `ExgSocketType`). The structure should contain values for the `target`, `type`, or `name` fields.

→ *unwrap*
If `true`, only an application that registered to receive the data type with the `exgUnwrap` flag set should be the target application. If `false`, the target application should be an application that registered with the `exgUnwrap` flag clear.

← *creatorIDP*
The creator ID of the application that should receive this object.

↔ *descriptionP*
The application's description from the registry, if any.

→ *descriptionSize*
The size of the `descriptionP` buffer.

Returns Returns one of the following error codes:

`errNone`	Success
`exgErrTargetMissing`	The `target` field contains a creator ID, but the application with that creator ID does not exist
`exgErrNoKnownTarget`	No application is registered to receive the data type

Comments The Exchange Manager uses this function to determine which application should be launched to receive incoming data. Applications and libraries may call this function as well.

`ExgGetTargetApplication` determines the target application by doing the following:

- If the `socketP->target` field contains a creator ID, the Exchange Manager searches the registry to see if an application is registered for that creator ID as the default application. If the registry does not contain an entry for the creator ID, it checks to see if the application identified by the creator ID is installed on this device. If an application is found for the `target`, that is the application returned.
- If the `socketP->type` field contains a MIME type, the Exchange Manager searches the registry for an application registered to receive objects of that type. If one is found, that is the application returned.

- If the `socketP->name` field contains a period (.), the portion after the last period is taken to be the file extension. The Exchange Manager searches the registry for an application registered to receive a file with the specified extension. If one is found, that is the application returned. If not, `exgErrNoKnownTarget` is returned.

If more than one application is registered for the target, type, or file extension, this function returns the one that is registered as the default. If no application is registered as the default, then a specific application is chosen. The Exchange Manager chooses this same application each time. That is, each time a file with a TXT extension is sent with no target or MIME type specified, the `ExgGetTargetApplication` returns the same application to handle the receipt.

Set the `unwrap` parameter to `true` if the object was sent as part of another object, such as a vStock object that was sent as an attachment to an e-mail message. In this case, the Exchange Manager searches for an application that registered to receive the target, the type, or the file extension of the vStock object with the `exgUnwrap` flag set. If an application is found, the vStock object is delivered, and the exchange library should discard the object that contained it (the e-mail message). If there is no application registered to receive the data with the `exgUnwrap` flag set, this function returns `exgErrNoKnownTarget`. In this case, the exchange library should call `ExgNotifyReceive()` again passing the entire e-mail message instead of just the vStock attachment.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `ExgSetDefaultApplication()`, `ExgNotifyPreview()`, `ExgNotifyReceive()`, `ExgRegisterDatatype()`

ExgNotifyGoto Function

Purpose Launches the target application using sysAppLaunchCmdGoTo.

Declared In ExgMgr.h

Prototype

```
Err ExgNotifyGoto (ExgSocketType *socketP,
    UInt16 flags)
```

Parameters

→ *socketP*
A socket identifying the object to deliver (see ExgSocketType). The goToCreator field contains the application to be launched, and the goToParams field contains data for the launch code's parameter block.

→ *flags*
Not currently used. Pass 0 for this parameter.

Returns Returns one of the following error codes:

errNone	Success or the goToCreator field is empty
dmErr... (one of the data manager error codes)	The specified application could not be found
memErrNotEnoughSpace	Not enough memory available to create the launch code's parameter block

Comments Exchange libraries call this function if they want to support immediate display of the received object. Applications do not call this function.

Most exchange libraries should call ExgNotifyGoto after the return from ExgNotifyReceive() so that the user can inspect the newly received data. If the exchange library is most often used by a single application that does not require the launch code, this call to ExgNotifyGoto can be skipped. For example, the SMS Library does not call ExgNotifyGoto. SMS messages are received by the SMS Messenger application, which does not launch upon receiving data.

ExgNotifyGoto only launches an application if one is specified in the goToCreator field and the noGoTo parameter is false. If a goToCreator is not specified, it is not considered an error. This gives the application a way to override the default behavior.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `ExgNotifyReceive()`, `ExgDisconnect()`

ExgNotifyPreview Function

Purpose Displays a preview in the exchange dialog of the data to be received.

Declared In `ExgMgr.h`

Prototype

```
Err ExgNotifyPreview (ExgPreviewInfoType *infoP)
```

Parameters

↔ *infoP*
An `ExgPreviewInfoType` structure containing information about the preview operation.

Returns Returns one of the following error codes:

`errNone`	Success
`exgErrNotSupported`	The exchange library doesn't support preview mode
`exgErrNoKnownTarget`	There is no application registered to receive the type of object

Other error codes depend on the application.

Comments This function performs the preview operation specified in the `op` field of the `infoP` parameter. The `ExgDoDialog()` function calls this function to show a data preview in the exchange dialog. Exchange libraries might want to call this function in certain circumstances. An application rarely calls this function, but it may do so if it displays its own dialog in response to the launch code `sysAppLaunchCmdExgAskUser`.

`ExgNotifyPreview` uses `ExgGetTargetApplication()` to determine the appropriate target application for this data and then launches that application with the launch code `sysAppLaunchCmdExgPreview`, passing `infoP` as the parameter block. The application responds to this launch code by accepting the connection, receiving the data from the exchange library, and depending on the operation requested, drawing the data into the `infoP->bounds` rectangle or returning it in the `infoP->string`

field, and then disconnecting. The `ExgDoDialog` function uses the returned information to draw the preview portion of the dialog.

If the preview data is a string, the `ExgNotifyPreview` provides a series of fallback strings that are used if the exchange library doesn't support preview or the application doesn't respond to the launch code. If the application fails to return a string, this function provides one of the following:

- the data's description from `socketP->description`
- the filename in `socketP->name`
- the target application's description as stored in the exchange registry
- the MIME type in `socketP->type`
- the file extension in `socketP->name`

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `ExgNotifyReceive()`, `ExgDisconnect()`

ExgNotifyReceive Function

Purpose Delivers an object to the appropriate application using the registry.

Declared In `ExgMgr.h`

Prototype

```
Err ExgNotifyReceive (ExgSocketType *socketP,
    UInt16 flags)
```

Parameters

↔ *`socketP`*
A pointer to the socket structure (see `ExgSocketType`).

→ *`flags`*
A bit field. Pass 0 or a combination of the following constants (OR the constants together to specify more than one):

`exgUnwrap`
The object being delivered should only be handled by an application that registered to receive it with the `exgUnwrap` flag set.

`exgNoAsk`
Do not ask the user to confirm receipt of data. If this constant is passed, the target application does not

receive the sysAppLaunchCmdExgAskUser launch code, and the Exchange Manager does not call ExgDoDialog() to display the user confirmation dialog.

`exgGet`
Specifies that this is a request for the application to send data rather than to receive data.

Returns Returns one of the following error codes:

`errNone`	Success
`exgErrTargetMissing`	The `target` field contains a creator ID, but the application with that creator ID does not exist
`exgErrNoKnownTarget`	No application is registered to receive the data type
`exgErrUserCancel`	The user cancelled the operation

Other error codes depend on the application that is launched.

Comments Exchange libraries call this function to initiate a receive operation on the receiving device. Applications do not call this function.

The `ExgNotifyReceive` function uses ExgGetTargetApplication() to determine which application should receive the data, then sends that application the appropriate launch codes.

If the `flags` parameter is 0, a receive operation is assumed. The `ExgNotifyReceive` function does the following:

1. It sends the application the sysAppLaunchCmdExgAskUser launch code.
2. If the application returns `exgAskDialog` or does not respond to the launch code, it calls ExgDoDialog(), which sends the application the sysAppLaunchCmdExgPreview launch code to have the application receive preview data for the dialog.
3. It sends the application the sysAppLaunchCmdExgReceiveData launch code to tell the application to receive the data.

If the `flags` field contains the `exgNoAsk` flag, the first and second steps are skipped.

If the `flags` field contains `exgGet`, this function is a request for data to send to the remote device, not a request to receive data from the remote device. In this case, `ExgNotifyReceive` launches the target application with the `sysAppLaunchCmdExgGetData` launch code.

If the `flags` field has the `exgUnwrap` bit set, it means that the object to be received was sent as part of another object, and it should only be sent to an application that registered to receive it with the `exgUnwrap` flag set. For example, if the exchange library receives an e-mail message with an attached vStock object, the exchange library may call `ExgNotifyReceive` with the `exgUnwrap` flag set and a socket that describes the vStock data type to see if there is an application that registered to receive it directly. If no application is registered to receive vStock objects with the `exgUnwrap` flag set, `ExgNotifyReceive` returns `exgErrNoKnownTarget`. The exchange library should then call `ExgNotifyReceive` again, but this time without the `exgUnwrap` flag and with a socket that describes the e-mail message data type. This second call sends the object to the application registered to receive the e-mail message rather than its vStock attachment. That application may extract the vStock attachment from the message and use the Local Exchange Library to send it to an application registered to receive vStock objects normally (without the `exgUnwrap` flag).

Compatibility

Implemented only if 3.0 New Feature Set is present. `ExgNotifyReceive` was a system use only function until the release of Palm OS 4.0.

If the 4.0 New Feature Set is *not* present, the `flags` parameter is not supported, so libraries cannot suppress the exchange dialog, send objects with attachments, or perform a get operation. These features are all added in the 4.0 New Feature Set. Also, if the 4.0 new feature set is not present, this function performs the equivalent of `ExgNotifyGoto()` after the application has returned from receiving data. Exchange libraries wishing to support this functionality should call `ExgNotifyGoto` immediately after calling `ExgNotifyReceive`.

See Also

`ExgNotifyPreview()`

ExgPut Function

Purpose Initiates the transfer of data to the destination device.

Declared In `ExgMgr.h`

Prototype

```
Err ExgPut (ExgSocketType *socketP)
```

Parameters

→ *`socketP`*

Pointer to the socket structure (see `ExgSocketType`). Specify either a value for the `libraryRef` field or a URL in the `name` field. `libraryRef` should be 0 if the `name` field contains a URL. The structure should also contain a value for the `target`, `type`, or `name` field.

Returns Returns one of the following error codes:

`errNone`	Success
`exgErrBadLibrary`	Couldn't find default exchange library
`exgMemError`	Not enough memory to initialize transfer
`exgErrNotEnoughPower`	The battery does not have enough power to perform the operation

Other error codes depend on the exchange library.

Comments Applications call this function to start a send operation.

If the connection does not already exist, this function establishes one. You must create and initialize an `ExgSocketType` structure containing information about the data to send and the destination application. All unused fields in the structure *must* be set to 0.

If no error is returned, this call *must* be followed by `ExgSend()`, to begin sending data, or `ExgDisconnect()`, to disconnect. You may need to call `ExgSend` multiple times to send all the data.

The socket's `libraryRef` field or the `name` field must identify the library that performs the transfer. The `libraryRef` field identifies the exchange library by its library reference number. The `name` field identifies the library by URL. The socket should also specify what data is being sent by providing values for at least one of the `target`, `name`, and `type` fields. Use of the `name` field is the most common method.

To provide users with a choice of transport mechanisms, the application can provide a URL that begins with a question mark (?). The Exchange Manager displays a dialog with a list of all exchange libraries that respond to URLs of the specified type. If only one exchange library is registered for this URL scheme, no dialog is displayed.

For example, many applications on Palm OS 4.0 or higher support a Send command. This command generates a URL with the prefix `exgSendPrefix` (see Predefined URL Prefixes). The Exchange Manager displays a dialog containing a list of libraries registered for that URL scheme. The user selects an exchange library, and that library's `ExgLibSend` function is called.

If the library is not specified by either URL or library reference number, the Exchange Manager by default uses the IR Library; however, if the `localMode` flag is set, the Local Exchange Library is used instead.

Compatibility Implemented only if 3.0 New Feature Set is present.

Support for identifying exchange libraries by URL is implemented only if 4.0 New Feature Set is present.

See Also `ExgDisconnect()`, `ExgSend()`, `ExgConnect()`

ExgReceive Function

Purpose Receives data from a remote device.

Declared In `ExgMgr.h`

Prototype

```
UInt32 ExgReceive (ExgSocketType *socketP,
    void *bufP, UInt32 bufLen, Err *err)
```

Parameters

→ *socketP*
A pointer to the socket structure (see `ExgSocketType`).

← *bufP*
A pointer to the buffer in which to receive the data.

→ *bufLen*
The number of bytes to receive.

← *err*
A pointer to an error code result.

Returns Returns the number of bytes actually received. A zero result indicates the end of the transmission.

An error code is returned in the address indicated by `err`. The error code `exgErrUserCancel` is returned if the user cancels the operation. The error code `exgErrNotSupported` is returned if the application calls this function during a preview and the exchange library does not have any more data available or does not support preview.

May display a fatal error message if the library reference number is not provided in `socketP`.

Comments Applications call this function in the following circumstances:

- In response to the `sysAppLaunchCmdExgReceiveData` launch code, following a successful call to `ExgAccept()`.
- In response to the `sysAppLaunchCmdExgPreview` launch code, following a successful call to `ExgAccept`.
- To receive requested data following a successful call to `ExgGet()`.

After receiving the data, applications call `ExgDisconnect()` to terminate the connection.

This function blocks the application until the end of the transmission or until the requested number of bytes has been received. However, exchange libraries can provide their own user interface that is shown during this call, is updated as necessary, and allows the user to cancel the operation in progress.

Compatibility Implemented only if 3.0 New Feature Set is present. Preview mode and `ExgGet()` are only supported if 4.0 New Feature Set is present.

See Also `ExgNotifyReceive()`

ExgRegisterDatatype Function

Purpose Registers an application to receive a specific type of data, or registers an exchange library to handle specific URL schemes.

Declared In `ExgMgr.h`

Prototype

```
Err ExgRegisterDatatype (UInt32 creatorID,
   UInt16 id, const Char *dataTypesP,
   const Char *descriptionsP, UInt16 flags)
```

Parameters

→ *creatorID*
The creator ID of the registering application or exchange library.

→ *id*
A registry ID constant identifying the type of the items being registered. See Registry ID Constants.

→ *dataTypesP*
Pointer to a tab-delimited, null-terminated string listing the items to register. (Use `"\t"` for the tab character.) To unregister, pass a `NULL` value. Each item in the string must be no more than `exgMaxTypeLength` characters. There can be no more than 16 types total.

NOTE: If specifying file extensions, do not include the period (.) that precedes the extension. If specifying URL prefixes, do not include the colon (:) at the end of the prefix.

→ *descriptionsP*
Pointer to a tab-delimited, null-terminated string that lists descriptions for the items in the `dataTypesP` parameter. (Use `"\t"` for the tab character.) Each description must be no longer than `exgMaxDescriptionLength`. Pass `NULL` to leave out the descriptions.

There must either be one description for all types or the number of descriptions must match the number of types.

The descriptions are used in dialogs displayed by Exchange Manager to identify applications or libraries.

→ *flags*
A bit field specifying registration options. Currently, only one bit is used: the unwrap bit. Pass the `exgUnwrap` constant to specify that the application is registering to receive objects of

this type directly if the object is sent as part of another object. For example, if a vStock object is sent as an attachment to an email message, the Exchange Manager should send the vStock object to this application directly rather than sending the message to the email application.

Returns Returns `errNone` if successful, `exgMemError` if there is not enough memory to save the registration info, or one of the data manager error codes (`dmErr`...).

Comments Both applications and exchange libraries use this function to register with the Exchange Manager to receive certain types of data.

Applications must register with the Exchange Manager to receive data objects that do not specifically target that application using the creator ID in the `target` field.

Exchange libraries register to receive data with certain URL schemes. If an exchange library is not registered to receive URLs, it only handles the receipt and sending of data if its library reference number is explicitly specified in the `ExgSocketType` structure. Otherwise, the IR Library handles all incoming data for which a library could not be found.

Both applications and libraries should register to receive data as soon as possible after they are installed and as soon as possible after a hard reset. For example, applications can call `ExgRegisterDatatype` in response to the `sysAppLaunchCmdSyncNotify` launch code, which they receive immediately after install. Exchange libraries implemented as applications can also use this strategy. Exchange libraries implemented as shared libraries should call `ExgRegisterDatatype` in their startup functions.

Make only one call to `ExgRegisterDatatype` per registry type. If you want to register to receive multiple items, use a tab character (`\t`) to separate the items. If you were to, for example, make one call to register for the DOC file extension and one call to register for the TXT extension, the second call overwrites the first. However, if you want to register with the `exgUnwrap` flag set, make one call without the `exgUnwrap` flag and one call with the `exgUnwrap` flag set. The application registered with the `exgUnwrap` flag set is stored in a different part of the registry.

Specify `exgRegExtensionID` to register to receive data that has a filename with a particular extension. For example, if your application wants to receive files with a TXT extension, it could register like this:

```
ExgRegisterDatatype(myCreator,
  exgRegExtensionID, "TXT", NULL, 0);
```

If the application wants to receive files with a TXT extension or with a DOC extension, it could register like this:

```
ExgRegisterDatatype(myCreator,
  exgRegExtensionID, "TXT\tDOC", NULL, 0);
```

Specify `exgRegTypeID` to register to receive data with a specific MIME type. For example, if your application wants to receive "setext" text files, it could register like this:

```
ExgRegisterDatatype(myCreator, exgRegTypeID,
"text/x-setext", NULL, 0);
```

Specify `exgRegCreatorID` to register to receive data targeted for a particular creator ID. For example, if your application wants to handle all data intended for the ToDo application, it could register like this:

```
Char toDoCreatorStr[5];
MemMove(toDoCreatorStr, sysFileCToDo, 4);
toDoCreatorStr[4] = chrNull;
ExgRegisterDatatype(myCreator, exgRegCreatorID,
  toDoCreatorStr, NULL, 0);
```

NOTE: To override one application's receipt of data, you need to also set your application as the default for this creator ID. See `ExgSetDefaultApplication()`.

Most exchange libraries will want to register for a unique URL scheme that identifies only that library, plus they should register for a more general scheme, such as the send scheme (`exgSendScheme`), which causes the library to be listed in the Send

With dialog when the user performs the Send command. The registry ID constant for URL prefixes is `exgRegSchemeID`.

```
ExgRegisterDatatype(myLibCreator,
  exgRegSchemeID, myScheme "\t" exgSendScheme,
  NULL, 0);
```

Registrations are active until a hard reset or until the application or library is removed. The registration information is preserved across a soft reset. When an application is removed, its registry information is also automatically removed from the registry, so there is not normally a need to unregister. If you want to unregister, you can call `ExgRegisterDatatype` with a `NULL` value for the `dataTypesP` parameter.

Multiple applications can be registered to receive the same type of data. If this is the case, the application that is registered as the default (using `ExgSetDefaultApplication()`) is the one that receives the data unless the exchange socket explicitly specifies another application should receive it. If there is no default specified, the Exchange Manager determines a default.

Multiple libraries may also be registered to receive the same type of URL. In this case, if the URL begins with a question mark (?), the Exchange Manager displays a dialog so that the user can select which exchange library to use. If the URL does not begin with a question mark, the exchange library registered as the default is used. If there is no default specified, the Exchange Manager determines a default.

Compatibility

Implemented only if 4.0 New Feature Set is present. `ExgRegisterDatatype` replaces the `ExgRegisterData()` function.

See Also

`ExgRegisterData()`, `ExgGetTargetApplication()`, `ExgPut()`, `ExgGetDefaultApplication()`, `ExgGetRegisteredApplications()`, `ExgGetRegisteredTypes()`

ExgRegisterData Function

Purpose Registers an application to receive a specific type of data. This function is deprecated and replaced with `ExgRegisterDatatype()`.

Declared In `ExgMgr.h`

Prototype

```
Err ExgRegisterData (UInt32 creatorID, UInt16 id,
    const Char *dataTypesP)
```

Parameters

→ *creatorID*
Creator ID of the registering application.

→ *id*
Registry ID identifying the type of the items being registered. Specify `exgRegExtensionID` or `exgRegTypeID`.

→ *dataTypesP*
Pointer to a tab-delimited, null-terminated string listing the items to register. (Use \t for the tab character.) These include file extensions or MIME types. To unregister, pass a `NULL` value.

Returns Returns `errNone` if successful, otherwise, one of the data manager error codes (`dmErr`...).

Comments Applications that wish to receive data from anything other than another Palm Powered™ handheld running the same application must use this function to register for the kinds of data they can receive. Call this function when your application is loaded on the device.

Compatibility This function corresponds to the Palm OS 3.5 version of `ExgRegisterDatatype()`. It is implemented only if 3.0 New Feature Set is present.

ExgRequest Function

Purpose Requests some data from an exchange library or an application using a URL.

Declared In `ExgMgr.h`

Prototype

```
Err ExgRequest (ExgSocketType *socketP)
```

Parameters → *`socketP`*

Pointer to the socket structure (see `ExgSocketType`). Specify a URL in the `name` field and a `libraryRef` of 0.

Returns Returns one of the following error codes:

`errNone`	Success
`exgErrBadLibrary`	Couldn't find default exchange library
`exgErrNotEnoughPower`	The device does not have enough power to perform the operation
`sysErrLibNotFound`	Couldn't find library or application to respond to URL

Other error codes depend on the exchange library or application.

Comments The `ExgRequest` function is similar to `ExgGet()` in that both are used to request data. The difference is that the application that calls `ExgGet` is always the application that receives the data. When you call `ExgRequest`, the application that receives the data is the application that is registered to receive it. For example, using `ExgRequest`, it is possible for one application to use the Exchange Manager to retrieve a vCard using any supported transport mechanism and have that data sent directly to the Address Book application instead of to the calling application.

The `socketP` passed to this function identifies the exchange library using a URL in the `name` field. The application must know beforehand the proper URL prefix for the exchange library with which it wants to connect. See Predefined URL Prefixes for a list of URL prefixes that the Exchange Manager provides.

If the provided URL begins with a question mark (?) and there are several exchange libraries registered for the specified URL scheme,

the Exchange Manager displays a dialog from which the user selects the appropriate transport mechanism.

If the Exchange Manager cannot find a library that is registered for the specified URL, it assumes that an application is registered to receive the URL, and it launches that application with the `sysAppLaunchCmdGoToURL` launch code.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `ExgGet()`, `ExgNotifyReceive()`

ExgSend Function

Purpose Sends data to the destination device.

Declared In `ExgMgr.h`

Prototype

```
UInt32 ExgSend (ExgSocketType *socketP,
    const void *bufP, UInt32 bufLen, Err *err)
```

Parameters

→ *socketP*
: A pointer to the socket structure (see `ExgSocketType`). A value must be provided for the `libraryRef` field. The structure should also contain values for the `target`, `type`, or `name` fields.

→ *bufP*
: A pointer to the data to send.

→ *bufLen*
: The number of bytes to send.

← *err*
: A pointer to an error code result.

Returns Returns the number of bytes actually sent, normally the same number as specified in `bufLen`. An error code is returned in the address indicated by `err`. The error code `exgErrUserCancel` is returned if the user cancels the operation.

May display a fatal error message if the `socketP` parameter does not contain a value for the `libraryRef` field.

Comments Call this function one or more times to send all the data, following a successful call to `ExgPut()`. After sending the data, call `ExgDisconnect()` to terminate the connection.

The exchange library may break large amounts of data into multiple packets or assemble small send commands together into larger packets, but the application will not be aware of these transport level details.

This function blocks the application until all the data is sent. However, the exchange library may provide its own user interface that is updated as necessary and allows the user to cancel the operation in progress.

Compatibility Implemented only if 3.0 New Feature Set is present.

See Also `ExgReceive()`, `ExgGet()`

ExgSetDefaultApplication Function

Purpose Sets the application that receives a specified type of data by default. This function also sets the default exchange library that handles particular URL schemes.

Declared In `ExgMgr.h`

Prototype

```
Err ExgSetDefaultApplication (UInt32 creatorID,
   UInt16 id, const Char *dataTypeP)
```

Parameters

→ *creatorID*
The creator ID of the application or library that should become the default for this type of data.

→ *id*
A registry ID constant identifying the type of data in `dataTypeP`. See Registry ID Constants.

→ *dataTypesP*
A pointer to a null-terminated string containing the desired type of data.

NOTE: If specifying a file extension, do not include the period (.) that precedes the extension. If specifying a URL prefix, do not include the colon (:) at the end of the prefix.

Returns Returns `errNone` upon success or `exgErrNoKnownTarget` if the specified application is not registered to receive the specified data type.

Comments This function sets the default application that receives data of a certain type when no target is specified and the default exchange library that handles URLs with a certain prefix if no library reference number is specified.

Palm™ strongly recommends that applications allow the user to determine which application should become the default recipient for a data type. To do so, an application can use `ExgGetRegisteredApplications()` to get the list of applications registered for the same type of data as it is, and then display a dialog listing those applications and allow the user to select it. Then it should call `ExgSetDefaultApplication` with the user-specified default.

If you call `ExgSetDefaultApplication` with an application or library that is already the default, this function has no effect.

An application can become the default for its own creator ID even if it has not specifically registered to receive its own creator ID. That is, suppose several applications are registered to receive objects targeted for the ToDo application's creator ID. The ToDo application itself is not registered for its own creator ID, as it is not necessary to do so. However, an application can use code like the following to set the ToDo application as the default for its own creator ID.

```
Char toDoCreatorStr[5];
MemMove(toDoCreatorStr, sysFileCToDo, 4);
toDoCreatorStr[4] = chrNull;
ExgSetDefaultApplication(sysFileCToDo,
  exgRegCreatorID, toDoCreatorStr);
```

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `ExgGetDefaultApplication()`, `ExgRegisterDatatype()`

Application-Defined Functions

ExgDBDeleteProcPtr Function

Purpose Handles the case where a database with an identical name already exists on the device.

Declared In `ExgMgr.h`

Prototype

```
Boolean (*ExgDBDeleteProcPtr) (const char *nameP,
   UInt16 version, UInt16 cardNo, LocalID dbID,
   void *userDataP)
```

Parameters

→ *nameP*
A pointer to the name of the identical database.

→ *version*
The version of the identical database.

→ *cardNo*
The card number of the identical database.

→ *dbID*
The database ID of the identical database.

→ *userDataP*
The `userDataP` parameter you passed to `ExgDBRead()`. If used, this parameter contains any application-specific data you find necessary. If the `ExgDBReadProcPtr()` function is implemented using Exchange Manager calls, this often contains the `ExgSocketType` structure.

Returns Return `true` to have the `ExgDBRead` function continue to read the database. Use this return value if you have deleted or moved the existing database or if you want the database to be overwritten. Return `false` to have `ExgDBRead` exit without reading the database.

Comments This function is called if the Data Manager can't create the incoming database because a database with the same name already exists. You should delete the existing database or take some other action, such as changing the database name. It is appropriate to prompt the user before choosing to delete or move the database.

ExgDBReadProcPtr Function

Purpose Reads in the database and pass it to `ExgDBRead()`.

Declared In `ExgMgr.h`

Prototype

```
Err (*ExgDBReadProcPtr) (void *dataP,
    UInt32 *sizeP, void *userDataP)
```

Parameters

← *dataP*

A pointer to a buffer where this function should place the database data. This buffer is allocated in the dynamic heap by `ExgDBRead`; you don't need to use `DmWrite()` when filling it.

↔ *sizeP*

The size of `dataP`. This value is set by `ExgDBRead` to the number of bytes it expects to receive in `dataP`. You must set this value to the number of bytes you return in `dataP` (if it's not the same).

→ *userDataP*

The `userDataP` parameter you passed to `ExgDBRead`. Pass the `ExgSocketType` structure if you implement this function using Exchange Manager calls.

Returns Return an error number, or `errNone` if there is no error. If this function returns an error, `ExgDBRead` deletes the database it was creating, cleans up any memory it allocated, then exits, returning the error passed back from this function.

Comments `ExgDBRead` is commonly used to receive a database from a beam or from some other transport mechanism. In this case, an appropriate implementation of this callback function is to call `ExgReceive()` as shown here:

```
Err MyReadDBProc (void *dataP, UInt32 *sizeP,
  void *userDataP)
{
  Err err = errNone;
  //userDataP contains ExgSocketType pointer.
  *sizeP =
    ExgReceive((ExgSocketType *)userDataP,
      dataP, *sizeP, &err);
  return err;
}
```

ExgDBWriteProcPtr Function

Purpose Writes out the database.

Declared In `ExgMgr.h`

Prototype

```
Err (*ExgDBWriteProcPtr) (const void *dataP,
    UInt32 *sizeP, void *userDataP)
```

Parameters

→ *dataP*
: A pointer to a buffer containing the database data, placed there by `ExgDBWrite()`.

↔ *sizeP*
: The number of bytes placed in `dataP` by `ExgDBWrite`. If you were unable to write out or send all of the data in this chunk, on exit, set `sizeP` to the number of bytes you did write.

→ *userDataP*
: The `userDataP` parameter you passed to `ExgDBWrite`. You can use it for application-specific data. Pass the `ExgSocketType` structure if you implement this function using Exchange Manager calls.

Returns Return an error number, or `errNone` if there is no error. If this function returns an error, `ExgDBWrite` closes the database it was reading, cleans up any memory it allocated, then exits, returning the error passed back from this function.

Comments `ExgDBWrite` is commonly used to write a database that is going to be beamed to another device (or sent through some other transport mechanism). In this case, an appropriate implementation of this callback function is to call `ExgSend()` as shown here:

```
Err MyWriteDBProc (void *dataP, UInt32 *sizeP,
  void *userDataP)
{
  Err err = errNone;
  //userDataP contains ExgSocketType pointer.
  *sizeP =
    ExgSend((ExgSocketType *)userDataP,
      dataP, *sizeP, &err);
  return err;
}
```

63

Exchange Library

The Exchange Library API described in this chapter and declared in `ExgLib.h` specifies a minimal set of functions that all exchange libraries must implement. This chapter is directed towards developers who use or create exchange libraries. Developers creating an exchange library should also read the Exchange Libraries chapter of the *Palm OS Programmer's Companion*, vol. II, *Communications*.

Exchange Library Functions

ExgLibAccept Function

Purpose Accept an incoming connection.

Declared In `ExgLib.h`

Prototype

```
Err ExgLibAccept(UInt16 libRefnum,
    ExgSocketType *exgSocketP)
```

Parameters

→ *`libRefnum`*
Reference number of this exchange library.

→ *`exgSocketP`*
A pointer to the socket structure (see `ExgSocketType`).

Returns Returns `errNone` if no error. `exgErrNotSupported` is returned if a preview is in progress and the exchange library does not support preview. Other error codes are defined by each exchange library.

Comments The Exchange Manager's `ExgAccept()` function simply calls `ExgLibAccept` in the exchange library identified by the `ExgSocketType` structure passed to `ExgAccept`. An application calls the Exchange Manager's `ExgAccept` function when:

- The application wants to initiate a connection to receive data, which it does in response to `sysAppLaunchCmdExgReceiveData`.

- The application wants to initiate a connection to receive a preview of the data, which it does in response to `sysAppLaunchCmdExgAskUser`.

Any implementation of `ExgLibAccept` should update any progress dialogs to indicate that data is being accepted (or received) into an application.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `ExgLibPut()`

ExgLibClose Function

Purpose Library-specific. Although this function is not called by the Exchange Manager, all shared libraries normally implement it.

Declared In `ExgLib.h`

Prototype

```
Err ExgLibClose (UInt16 libRefnum)
```

Parameters

→ *libRefnum*
Reference number of this exchange library.

Comments Exchange libraries are free to implement this function for internal or external use. The Exchange Manager does not call it.

ExgLibConnect Function

Purpose Open a connection in preparation for sending or receiving objects.

Declared In `ExgLib.h`

Prototype

```
Err ExgLibConnect(UInt16 libRefNum,
   ExgSocketType *exgSocketP)
```

Parameters

→ *libRefNum*
Reference number of this exchange library.

↔ *exgSocketP*
A pointer to an `ExgSocketType` structure identifying the socket through which objects will be sent or received.

Returns Returns `errNone` if no error. If `ExgLibConnect` is not supported by this library, this function returns `exgErrNotSupported`; if its

use is optional, `errNone` is returned. Other error codes are defined by each exchange library.

Comments

The Exchange Manager may call this function to initiate a connection for sending multiple objects or for performing two-way communications. Some exchange libraries support sending multiple objects but do not support this call. See "Sending Multiple Objects" on page 16 of *Palm OS Programmer's Companion*, vol. II, *Communications* for more information.

Not all exchange libraries support this operation. In this case, the first call to `ExgLibPut()` must clean up after itself before returning an error. The exchange library should not expect to get an `ExgLibDisconnect()` call.

If `ExgLibConnect` is supported and an application calls `ExgConnect()`, the exchange library should delay any cleanup until `ExgLibDisconnect`, unless `ExgLibConnect` returns an error, in which case it should clean up after itself. The library can expect to get an `ExgLibDisconnect` call if it returns `errNone` from `ExgLibConnect`. If the application does not call `ExgConnect`, the first call to `ExgLibPut` must clean up after itself before returning an error and the exchange library should not expect to get an `ExgLibDisconnect` call.

The Exchange Manager's `ExgConnect` function calls `ExgLibConnect` in the exchange library identified by the `ExgSocketType` structure passed to `ExgConnect`. If `ExgLibConnect` is implemented to return `exgErrNotEnoughPower`, the Exchange Manager puts up an alert, so there is no need for the exchange library to do so. Other error codes are not treated specially by the Exchange Manager; the exchange library must put up its own alerts when appropriate.

The exchange library may prompt the user for addressing information.

Compatibility

Implemented only if 4.0 New Feature Set is present.

ExgLibControl Function

Purpose Supply information about the exchange library.

Declared In `ExgLib.h`

Prototype

```
Err ExgLibControl(UInt16 libRefNum, UInt16 op,
   void *valueP, UInt16 *valueLenP)
```

Parameters

→ *libRefNum*
Reference number of this exchange library.

→ *op*
The operation to perform.

↔ *valueP*
An operation specific parameter. See comments below.

↔ *valueLenP*
An operation specific parameter. See comments below.

Returns Returns `errNone` if no error. `exgErrNotSupported` is returned if the specified operation is not supported by the exchange library. `exgErrBadParam` is returned if the parameters (`valueP` and `valueLenP`) are not appropriate for the operation. Additional error codes are defined by each exchange library.

Comments `ExgLibControl` is a general purpose function that performs various minor operations based upon a selector. `ExgMgr.h` defines three selectors: `exgLibCtlGetTitle`, which should be supported by all exchange libraries, `exgLibCtlGetVersion`, and `exgLibCtlGetPreview`. Additional library-specific selectors should be numbered starting at `exgLibCtlSpecificOp` (0x8000).

Upon receiving `exgLibCtlGetTitle`, the exchange library must return a library title suitable for use in Exchange Manager dialogs. Exchange libraries that are built as applications should generally return their `'tAIN'` resource as their title. Be sure that the title returned through `valueP` honors the maximum length specified in `valueLenP`.

An exchange library that implements the Palm OS 4.0 version of the Exchange Library API needn't do anything special upon receiving `exgLibCtlGetVersion`; it should simply return `exgErrNotSupported`. Otherwise, set `valueP` to a two-byte value indicating the API version number and return `errNone`. Note

that a version number of zero corresponds to the Palm OS 4.0 version of the Exchange Library API.

The `exgLibCtlGetPreview` operation is used by the Exchange Manager to determine whether a given exchange library supports preview. The Exchange Manager assumes that an exchange library supports preview if the exchange library doesn't implement this operation. To indicate that a library does not support preview, set `valueP` to `false` and return `errNone`.

The Exchange Manager's `ExgControl()` function simply calls `ExgLibControl` in the exchange library identified by the `ExgSocketType` structure passed to `ExgControl`.

Compatibility Implemented only if 4.0 New Feature Set is present.

ExgLibDisconnect Function

Purpose Disconnect a connection made with `ExgLibConnect()`, `ExgLibAccept()`, `ExgLibPut()`, or `ExgLibGet()`.

Declared In `ExgLib.h`

Prototype

```
Err ExgLibDisconnect(UInt16 libRefnum,
   ExgSocketType *exgSocketP, Err error)
```

Parameters

→ *libRefnum*
Reference number of this exchange library.

→ *exgSocketP*
A pointer to an `ExgSocketType` structure identifying the socket connection to be disconnected.

→ *error*
The current error state. Used to indicate why the connection is being broken, for example, user cancel or out of memory.

Returns Typically the same error code passed in. However, this function may return an error even if `errNone` is passed in.

Comments `ExgLibDisconnect` may be used to finish reading the data during a preview; in this case, the connection, if any, is not shut down.

Applications call `ExgLibDisconnect` when all data has been sent or the application wants to stop the send process. If the send data process is not completed, the caller should pass an error parameter

indicating why the operation was stopped. `ExgLibDisconnect` is responsible for completing the operation and closing any communication ports if necessary. If data was buffered for sending in `ExgLibSend`, then the disconnect process may actually perform the entire transmit operation. It is important to note that the `ExgLibDisconnect` function can be called for an `ExgLibPut`, `ExgLibAccept` or `ExgLibGet` function. So it is equally important to keep track of the current operation in the `ExgSocketType`. If dialogs are displayed, this function must update them as appropriate. If there are errors, `ExgLibDisconnect` should display them (if allowed by the application).

If `ExgLibConnect()` is not supported, the first call to `ExgLibPut()` must clean up after itself before returning an error and the exchange library should not expect to get an `ExgLibDisconnect` call.

If `ExgLibConnect` is supported and an application calls `ExgConnect()`, the exchange library should delay any cleanup until `ExgLibDisconnect`, unless `ExgLibConnect` returns an error, in which case it should clean up after itself. The library can expect to get an `ExgLibDisconnect` call if it returns `errNone` from `ExgLibConnect`. If the application does not call `ExgConnect`, the first call to `ExgLibPut` must clean up after itself before returning an error and the exchange library should not expect to get an `ExgLibDisconnect` call.

The Exchange Manager's `ExgDisconnect()` function simply calls `ExgLibDisconnect` in the exchange library identified by the `ExgSocketType` structure passed to `ExgDisconnect`.

Compatibility Implemented only if 4.0 New Feature Set is present.

ExgLibGet Function

Purpose Establish a connection and request data from a remote device.

Declared In `ExgLib.h`

Prototype

```
Err ExgLibGet(UInt16 libRefNum,
    ExgSocketType *exgSocketP)
```

Parameters

→ *`libRefNum`*
Reference number of this exchange library.

↔ *exgSocketP*
A pointer to the socket structure (see ExgSocketType).

Returns Returns `errNone` if no error. `exgErrNotSupported` is returned if this operation is not supported by this library. Other error codes are defined by each exchange library.

Comments `ExgLibGet` informs the library that it should make a connection to the remote device and request information from it. When an exchange library's `ExgLibGet` function is called, it should fetch the requested data and prepare to deliver it when the application calls `ExgReceive()`. After `ExgLibReceive()` is called, possibly more than once, `ExgLibDisconnect()` follows.

The Exchange Manager's `ExgGet()` function calls `ExgLibGet` in the exchange library identified in the `ExgSocketType` structure passed to `ExgGet`. If an exchange library's implementation of `ExgLibGet` returns `exgErrNotEnoughPower`, the Exchange Manager puts up an alert, so there is no need for the exchange library to do so. Other error codes are not treated specially by the Exchange Manager; an exchange library must put up its own alerts when appropriate.

Compatibility Implemented only if 4.0 New Feature Set is present.

ExgLibHandleEvent Function

Purpose Handle exchange-library-specific events.

Declared In `ExgLib.h`

Prototype

```
Boolean ExgLibHandleEvent(UInt16 libRefnum,
    void *eventP)
```

Parameters → *libRefnum*
Reference number of this exchange library.

→ *eventP*
The event to handle.

Returns Returns `true` if the event was handled.

Comments Exchange libraries are free to implement this function for internal or external use. The `Exchange Manager` does not call it.

ExgLibOpen Function

Purpose Library-specific. Although this function is not called by the Exchange Manager, all shared libraries normally implement it.

Declared In `ExgLib.h`

Prototype

```
Err ExgLibOpen (UInt16 libRefnum)
```

Parameters

→ *libRefnum*
Reference number of this exchange library.

Comments Exchange libraries are free to implement this function for internal or external use. The `ExgLibDisconnect` does not call it.

ExgLibPut Function

Purpose Signals the start of an object to be transferred to the destination device.

Declared In `ExgLib.h`

Prototype

```
Err ExgLibPut(UInt16 libRefnum,
   ExgSocketType *exgSocketP)
```

Parameters

→ *libRefnum*
Reference number of this exchange library.

← *exgSocketP*
Pointer to the socket structure (see `ExgSocketType`).

Returns Returns `errNone` if no error. Error codes are defined by each exchange library.

Comments Opens a connection if necessary. The actual data should be sent using `ExgLibSend()` after which the connection should be shut down with `ExgLibDisconnect()`. The exchange library may prompt the user for addressing information.

The first time this library is called, it may be necessary to allocate global variables and perform other initialization steps. It is usually a good idea to keep any state information about the open connection in the `socketRef` field of the `exgSocketP` structure. This data can then be passed to subsequent operations on that socket.

`ExgLibPut` should then check if the `socketRef` field of `exgSocketP` has been initialized. In some cases, an application

may already have filled `socketRef` with addressing information. The use of `socketRef` is entirely up to the exchange library. If `socketRef` is empty, the exchange library needs to fill in any addressing information. In order to do this, the exchange library needs to open its own dialog asking the user for whatever addressing information would be appropriate.

Exchange libraries are responsible for any validation of data entered in the addressing dialog and may use Address Book lookup or other system features to improve the user experience.

Once the user has completed addressing and confirmed the dialog, the exchange library should, in general, call the Progress Manager to open a progress dialog that remains open during the entire put operation. The progress dialog should not be opened if the `noStatus` option was passed or if the transaction is in asynchronous mode.

If the exchange library is displaying dialogs, it must also look for events and pass them to the Progress Manager.

Other operations within `ExgLibPut` depend on the exchange library. The exchange library may open communications ports and establish remote links at this time. Or it may just open a stream for buffering data until a later operation. If any errors occur in this process, the exchange library is responsible for removing any progress dialogs and returning an error. If displaying progress, the exchange library may also need to display an error using the Progress Manager before returning.

NOTE: The progress dialog is converted to an error dialog and waits until the user dismisses it. This may occur in `ExgLibPut` or later in `ExgLibDisconnect()`. It depends on whether the connection was made in `ExgLibPut` or whether it used an existing connection made by a previous call to `ExgLibConnect` or `ExgLibPut`.

The first call to `ExgLibPut()` must clean up after itself before returning an error if one of the following conditions exists:

- If `ExgLibConnect()` is not supported.

- If `ExgLibConnect` is supported and an application does not call `ExgLibConnect`.

The Exchange Manager's `ExgPut()` function calls `ExgLibPut` in the exchange library identified by the `ExgSocketType` structure passed to `ExgPut`. If `ExgLibPut` returns `exgErrNotEnoughPower`, the Exchange Manager puts up an alert, so there is no need for the exchange library to do so. Other error codes are not treated specially by the Exchange Manager; an exchange library must put up its own alerts when appropriate.

Compatibility Implemented only if 4.0 New Feature Set is present.

ExgLibReceive Function

Purpose Receive data from a remote device.

Declared In `ExgLib.h`

Prototype

```
UInt32 ExgLibReceive(UInt16 libRefNum,
   ExgSocketType *exgSocketP, void *bufP,
   UInt32 bufSize, Err *errP)
```

Parameters

→ *libRefNum*
Reference number of this exchange library.

↔ *exgSocketP*
A pointer to the socket structure (see `ExgSocketType`).

→ *bufP*
A pointer to a buffer into which the data is put.

→ *bufSize*
The size of the buffer in bytes.

← *errP*
The error code result: `errNone` if no error. `exgErrNotSupported` is returned if used during a preview and the exchange library does not support preview or if there appears to be more data available but `ExgLibReceive` cannot obtain it.Error codes are defined by each exchange library.

Returns The number of bytes received. Returns 0 if the object is complete. `ExgLibReceive` blocks until at least one byte is available or the object is complete.

Comments Use after `ExgLibGet()` or `ExgLibAccept()` to receive the contents of the object. May be used after `ExgLibAccept` to examine the contents during a preview.

`ExgLibReceive` must update any progress dialogs to indicate that data is being received. The `ExgLibReceive` should fill the buffer passed as much as possible and return the actual number of bytes that were stored in the buffer. `ExgLibReceive` must block for at least one byte if the stream is not complete. Returning zero bytes indicates the end of the data.

The Exchange Manager's `ExgReceive()` function simply calls `ExgLibReceive` in the exchange library identified by the `ExgSocketType` structure passed to `ExgReceive`.

Compatibility Implemented only if 4.0 New Feature Set is present.

ExgLibRequest Function

Purpose Requests an object using a URL, then has the Exchange Manager send it to the default application registered for the object's type.

Declared In `ExgLib.h`

Prototype

```
Err ExgLibRequest(UInt16 libRefNum,
    ExgSocketType *socketP)
```

Parameters

→ *libRefNum*
Reference number of this exchange library.

↔ *socketP*
Pointer to the socket structure (see `ExgSocketType`). The `name` field contains the URL of the object being requested.

Returns Returns `errNone` if no error. `exgErrNotSupported` is returned if the exchange library does not support this operation. Other error codes are defined by each exchange library.

Comments The exchange library may prompt the user for addressing information. The socket's `name` field may contain a URL with any of the schemes for which the exchange library registered. Not all exchange libraries support this operation.

NOTE: This function is often used to tickle an exchange library to make it check to see if there are new messages. These messages are then delivered as usual. So there may or may not be a specific object being requested.

The Exchange Manager's `ExgRequest()` function calls `ExgLibRequest` in the exchange library identified by the `ExgSocketType` structure passed to `ExgRequest`. If `ExgLibRequest` returns `exgErrNotEnoughPower`, the Exchange Manager puts up an alert, so there is no need for the exchange library to do so. Other error codes are not treated specially by the Exchange Manager; an exchange library must put up its own alerts when appropriate.

See Also ExgRegisterDatatype()

ExgLibSend Function

Purpose Send data for an object to a destination device.

Declared In `ExgLib.h`

Prototype

```
UInt32 ExgLibSend(UInt16 libRefNum,
   ExgSocketType *exgSocketP, const void *bufP,
   UInt32 bufLen, Err *errP)
```

Parameters

→ *libRefNum*
Reference number of this exchange library.

← *exgSocketP*
A pointer to the socket structure (see `ExgSocketType`).

→ *bufP*
A pointer to a buffer containing the data to send.

→ *bufLen*
The number of bytes to send.

← *errP*
The error code result. Error codes are defined by each exchange library

Returns Returns the number of bytes sent.

Comments Applications call `ExgSend()` after `ExgPut()` and in response to `sysAppLaunchCmdExgGetData`.

This function blocks until all bytes to be sent are actually sent, or until an error occurs (such as device full).

`ExgLibSend` may be called any number of times with varying size buffers to transmit information. If dialogs are being displayed, `ExgLibSend` must keep them updated (perhaps with animation or progress information). `ExgLibSend` must also check for events and let the Progress Manager handle them.

The Exchange Manager's `ExgSend` function simply calls `ExgLibSend` in the exchange library identified by the `ExgSocketType` structure passed to `ExgSend`.

Compatibility Implemented only if 4.0 New Feature Set is present.

ExgLibSleep Function

Purpose The device is going to sleep.

Declared In `ExgLib.h`

Prototype

```
Err ExgLibSleep(UInt16 libRefnum)
```

Parameters

→ *libRefnum*
Reference number of this exchange library.

Returns Returns `errNone` if no error. Error codes are defined by each exchange library.

Comments The device goes into sleep mode when the user turns the device off, the auto-off timer expires, or power is low. All shared libraries must implement `ExgLibSleep`; however, no processing is required and simply returning `errNone` is enough.

IMPORTANT: Libraries must return from this function quickly to allow sufficient time for emergency shutdown situations (removal of batteries, for example).

ExgLibWake Function

Purpose The device is waking up.

Declared In `ExgLib.h`

Prototype

```
Err ExgLibWake(UInt16 libRefnum)
```

Parameters → *`libRefnum`*
Reference number of this exchange library.

Returns Returns `errNone` if no error. Error codes are defined by each exchange library.

Comments The device wakes up when the user turns the device on. All shared libraries must implement `ExgLibWake`, although no processing is required; it is enough to simply return `errNone`.

64

IR Library

The IR (InfraRed) library is a shared library that provides a direct interface to the IR communications capabilities of the Palm OS®. This chapter provides reference material for the IR library API:

- IR Library Data Structures
- IR Library Constants
- IR Stack Callback Events
- IR Library Functions
- IAS Functions and Macros
- Application-Defined Functions

The header file `irlib.h` declares the IR library API. For more information on the IR library, see the chapter "Beaming (Infrared Communication)" in the *Palm OS Programmer's Companion*, vol. II, *Communications*.

IR Library Data Structures

This section lists some of the more important data types used by the IR library functions.

IrConnect Struct

Purpose The `IrConnect` structure is used to manage an IrLMP or Tiny TP connection.

Prototype

```
typedef struct _hconnect {
  UInt8 lLsap;
  UInt8 rLsap;
  UInt8 flags;
  UInt8 reserved;
  IrCallBack callBack;
  IrPacket packet;
```

```
  ListEntry packets;
  UInt16 sendCredit;
  UInt8 availCredit;
  UInt8 dataOff;
} _hconnect;
```

Fields

`lLsap`
: The local LSAP on which this connection listens.

`rLsap`
: The LSAP assigned to the remote side.

`flags`
: For system use only.

`reserved`
: Reserved for future use.

`callBack`
: Pointer to callback function. For system use only.

`packet`
: For system use only.

`packets`
: List of packets to send.

`sendCredit`
: Amount of credit from peer.

`availCredit`
: Amount of credit to give to peer.

`dataOff`
: Amount of data less than IrLAP size.

IrPacket Struct

Purpose The `IrPacket` structure is used for sending IrDA packets.

Prototype

```
typedef struct _IrPacket {
  ListEntry node;
  UInt8 *buff;
  UInt16 len;
  IrConnect *origin;
  UInt8 headerLen;
  UInt8 header[14];
  UInt8 reserved;
} IrPacket;
```

Fields

`node`
: For system use only.

`buff`
: Pointer to the send data buffer.

`len`
: Number of bytes in data buffer.

`origin`
: Pointer to connection that owns the packet. For system use only.

`headerLen`
: Number of bytes contained in the header. For system use only.

`header`
: Storage for header. For system use only.

`reserved`
: Reserved for future use.

IMPORTANT: The `node` field must be the first field in the structure. It is used internally by the stack.

IrIASObject Struct

Purpose

The `IrIASObject` structure is used as storage for an IAS object managed by the local IAS server. An object of this type is passed as the `obj` parameter to the `IrIAS_Add()` function.

Prototype

```
typedef struct _IrIasObject {
  UInt8 *name;
  UInt8 len;
  UInt8 nAttribs;
  IrIasAttribute *attribs;
} IrIasObject;
```

Fields

`name`
: Pointer to name of object.

`len`
: Length of object name.

`nAttribs`
: Number of attributes.

`attribs`
: Pointer to an array of attributes.

IrIasQuery Struct

Purpose

The `IrIasQuery` structure is used to perform IAS queries. The `IrIasQuery` object is passed as the `token` parameter to functions such as `IrIAS_Query()` and `IrIAS_Next()`.

Prototype

```
typedef struct _IrIasQuery {
  UInt8 queryLen;
  UInt8 reserved;
  UInt8 *queryBuf;
  UInt16 resultBufSize;
  UInt16 resultLen;
  UInt16 listLen;
  UInt16 offset;
  UInt8 retCode;
  UInt8 overFlow;
  UInt8 *result;
  IrIasQueryCallBack callBack;
} _IrIasQuery;
```

Fields

`queryLen`
: Total length of the query.

`reserved`
: Reserved for future use.

`queryBuf`
: Pointer to buffer containing the query.

`resultBufSize`
: Size of the result buffer.

`resultLen`
: Actual number of bytes in the result buffer.

`listLen`
: Number of items in the result list.

`offset`
: Offset into the results buffer.

`retCode`
: Return code of operation.

`overFlow`
: Set to `true` if result exceeded result buffer size.

`result`
: Pointer to buffer containing result.

`callBack`
: Pointer to query callback function.

IrCallbackParms Struct

Purpose

The `IrCallbackParms` structure is used to pass information from the stack to the upper layer of the stack (application). Not all fields are valid at any given time. The type of event determines which fields are valid. The `IrCallbackParms` object is passed as the second parameter to the `IrCallback` function.

Prototype

```
typedef struct {
  IrEvent event;
  UInt8 reserved1;
  UInt8 *rxBuff;
  UInt16 rxLen;
  IrPacket *packet;
  IrDeviceList *deviceList;
  IrStatus status;
  UInt8 reserved2;
} IrCallBackParms;
```

Fields

`event`
Event causing the callback.

`reserved1`
Reserved for future use.

`rxBuff`
Received data buffer.

`rxLen`
Length of data in received buffer.

`packet`
Pointer to packet being returned.

`deviceList`
Pointer to discovery device list.

`status`
Status of stack.

`reserved2`
Reserved for future use.

IrStatsType Struct

Purpose

The `IrStatsType` structure defines performance statistics for the IR Library. Use the `ExgControl()` function with an `irGetStatistics` operation to retrieve these statistics. See IR Control Constants for more information.

Prototype

```
typedef struct {
  UInt16 recLineErrors;
  UInt16 crcErrors;
} IrStatsType;
```

Fields

`recLineErrors`
The number of serial errors since the library opened.

`crcErrors`
The number of CRC errors since the library opened.

IR Library Constants

IR Control Constants

The IR control constants define operations that the IR Exchange Library can perform. You pass these constants as the operation parameter to `ExgControl()`. The following table lists the operation constants, the data that should be passed as the `valueP` parameter to `ExgControl`, and what operation is performed in response.

Operation Constant	value Data Type	Description
`irGetScanningMode`	`Boolean`. Output only.	Returns `true` in `*valueP` if beaming is enabled or `false` if beaming is disabled.
`irGetStatistics`	`IrStatsType`. Output only.	Returns performance statistics.
`irRestoreScanning`	None	Re-enables beaming after an `irSuppressScanning` operation. This operation keeps track of the number of requests that beaming be disabled and re-enables beaming only when the count reaches 0. This operation differs from `irSetScanningMode` in that it does not update the saved preferences.

Operation Constant	value Data Type	Description
`irSetScanningMode`	`Boolean`. Input only.	Enables or disables beaming. This operation modifies the saved preferences database, which is back up during a HotSync® operation. Because of this, beaming may remain disabled after a reset if you use this operation to disable it. If you want to temporarily disable beaming use `irSuppressScanning` and `irRestoreScanning` instead.
`irSetBaudMask`	`UInt16` containing a mask of the `irOpenOptSpeed`... constants defined in `IrLib.h`. Input only.	Sets the possible baud rates that the IR Library will use to those specified in `*valueP`. OR the `irOpenOptSpeed`... constants together to specify more than one. The default rate is 0, which causes the baud rate to be determined by the hardware. This operation is sometimes useful for debugging connections. Generally, you should set all bits up to the fastest rate you want to allow. To reset, use this operation again and pass 0 in `*valueP`. If you change the baud rate, your changes are until the device is reset or you perform this operation again.

Operation Constant	value Data Type	Description
`irSetSerialMode`	`Boolean`. Input only.	If the specified value is `true`, the IR Library uses the serial port instead of the infrared port until the device is reset. This option is useful for debugging. You can run your application in POSE and use the IR Library to communicate with a device connected in the cradle.
`irSetSupported`	`Boolean`. Input only.	If `true`, IR is supported on this device. If `false`, IR is not supported. You can use this constant to disable the unsupported dialog that normally displays when a beam is attempted and no IR support is available.
`irSuppressScanning`	None	Temporarily disables beam receive. This operation keeps track of the number of requests that beaming be disabled and re-enables beaming (through `irRestoreScanning`) only when the count reaches 0. This operation differs from `irSetScanningMode` in that it does not update the saved preferences.

IR Stack Callback Events

The IR stack calls the application by way of a callback function stored in each `IrConnect` structure. The callback function is called with a pointer to the `IrConnect` structure and a pointer to a parameter structure. The parameter structure contains an `event` field, which indicates the reason the callback is called, and other parameters, which have meaning based on the event.

The meaning of the events is described in the following sections.

LEVENT_DATA_IND

Data has been received. The received data is accessed using fields `rxBuff` and `rxLen`.

LEVENT_DISCOVERY_CNF

Indicates the completion of a discovery operation. The field `deviceList` points to the discovery list.

LEVENT_LAP_CON_CNF

The requested IrLAP connection has been made successfully. The callback function of all bound `IrConnect` structures is called.

LEVENT_LAP_CON_IND

Indicates that the IrLAP connection has come up. The callback of all bound `IrConnect` structures is called.

LEVENT_LAP_DISCON_IND

Indicates that the IrLAP connection has gone down. This means that all IrLMP connections are also down. A callback with event `LEVENT_LM_CON_IND` is not given. The callback function of all bound `IrConnect` structures is called.

LEVENT_LM_CON_CNF

The requested IrLMP/Tiny TP connection has been made successfully. Connection data from the other side is found using fields `rxBuff` and `rxLen`.

LEVENT_LM_CON_IND

Other device has initiated a connection. `IrConnectRsp` should be called to accept the connection. Any data associated with the connection request can be found using fields `rxBuff` and `rxLen`, data pointer and length, respectively.

LEVENT_LM_DISCON_IND

The IrLMP/Tiny TP connection has been disconnected. Any data associated with the disconnect indication can be found using fields `rxBuff` and `rxLen`, data pointer and length, respectively.

LEVENT_PACKET_HANDLED

A packet is being returned. A pointer to the packet exists in field `packet`.

LEVENT_STATUS_IND

Indicates that a status event from the stack has occurred. The `status` field indicates the status generating the event. Possible status values are as follows:

- `IR_STATUS_NO_PROGRESS` which means that IrLAP has no progress for 3 seconds threshold time (for example, the beam is blocked).
- `IR_STATUS_LINK_OK` which indicates that the no progress condition has cleared.
- `IR_STATUS_MEDIA_NOT_BUSY` which indicates that the IR media has transitioned from busy to not busy.

LEVENT_TEST_CNF

Indicates that a TEST command has completed. The `status` field indicates if the test was successful.

- `IR_STATUS_SUCCESS` indicates that the operation was successful and the data in the test response can be found by using the `rxBuff` and `rxLen` fields.
- `IR_STATUS_FAILED` indicates that no TEST response was received. The packet passed to perform the test command is passed back in the `packet` field and is now available (no separate packet handled event occurs).

LEVENT_TEST_IND

Indicates that a TEST command frame has been received. A pointer to the received data is in `rxBuff` and `rxLen`. A pointer to the packet that is sent in response to the test command is in the `packet` field. The packet is currently set up to respond with the same data sent in the command TEST frame. If different data is desired as a response, then you need to modify the packet structure. This event is sent to the callback function in all bound `IrConnect` structures. The IAS connections ignore this event.

IR Library Functions

IrAdvanceCredit Function

Purpose Advances the credit to the other side of the connection.

Declared In `IrLib.h`

Prototype

```
void IrAdvanceCredit (IrConnect *con,
    UInt8 credit)
```

Parameters

→ *con*
Pointer to `IrConnect` structure representing connection to which credit is advanced.

→ *credit*
Amount of credit to advance.

Returns Returns nothing.

Comments The credit passed by this function is added to the existing available credit, which must not exceed 127. This function only makes sense for a Tiny TP connection.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrBind Function

Purpose Obtains a local LSAP selector and registers the connection with the protocol stack.

Declared In `IrLib.h`

Prototype

```
IrStatus IrBind (UInt16 refNum, IrConnect *con,
    IrCallBack callBack)
```

Parameters

→ *refnum*
IR library refNum.

↔ *con*
Pointer to `IrConnect` structure.

→ *callBack*
Pointer to a callBack function that handles the indications and confirmation from the protocol stack.

Returns `IR_STATUS_SUCCESS` means the operation completed successfully. The assigned LSAP can be found in `con->lLsap`.

`IR_STATUS_FAILED` means the operation failed for one of the following reasons:

- `con` is already bound to the stack.
- There is no room in the connection table.

Comments The `IrConnect` structure is re-initialized. Any values stored in the structure are lost. The assigned LSAP is returned in the `lLsap` field of `con`. The type of the connection is set to IrLMP. The `IrConnect` must be bound to the stack before it can be used.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrClose Function

Purpose Closes the IR library. This releases the global memory for the IR stack and any system resources it uses. This must be called when an application is done with the IR library.

Declared In `IrLib.h`

Prototype

```
Err IrClose (UInt16 refnum)
```

Parameters

→ *refnum*
IR library refNum.

Returns Returns 0 if successful.

Comments Do not call this function unless the call to `IrOpen()` was successful.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrConnectIrLap Function

Purpose Starts an IrLAP connection.

Declared In `IrLib.h`

Prototype

```
IrStatus IrConnectIrLap (UInt16 refNum,
   IrDeviceAddr deviceAddr)
```

Parameters

→ *refnum*
IR library refNum.

→ *deviceAddr*
32-bit address of device to which connection should be made.

Returns `IR_STATUS_PENDING` means the operation started successfully; the result is returned by way of a callback.

`IR_STATUS_MEDIA_BUSY` means the operation failed because the media is busy. Media busy is caused by one of the following reasons:

- Other devices are using the IR medium.
- An IrLAP connection already exists.
- A discovery process is in progress.

Comments The result is signaled to all bound `IrConnect` structures by way of the callback function. The callback event is `LEVENT_LAP_CON_CNF` if successful or `LEVENT_LAP_DISCON_IND` if unsuccessful.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrConnectReq Function

Purpose Requests an IrLMP or Tiny TP connection.

Declared In `IrLib.h`

Prototype

```
IrStatus IrConnectReq (UInt16 refNum,
   IrConnect *con, IrPacket *packet,
   UInt8 credit)
```

Parameters

→ *refnum*
IR library refNum.

→ *con*
Pointer to `IrConnect` structure for handling the connection. The `rLsap` field must contain the LSAP selector for the peer on the other device. Also the type of the connection must be set. Use `IR_SetConTypeLMP` to set the type to an IrLMP connection or `IR_SetConTypeTTP` to set the type to a Tiny TP connection.

→ *packet*
Pointer to a packet that contains connection data. Even if no connection data is needed, the packet must point to a valid `IrPacket` structure. The packet is returned by way of the callback function with the `LEVENT_PACKET_HANDLED` event if no errors occur. The maximum size of the packet is `IR_MAX_CON_PACKET` for an IrLMP connection or `IR_MAX_TTP_CON_PACKET` for a Tiny TP connection.

→ *credit*
Initial amount of credit advanced to the other side. Must be less than 127. It is ANDed with 0x7f, so if it is greater than 127, unexpected results occur. This parameter is ignored if the connection is an IrLMP connection.

Returns `IR_STATUS_PENDING` means the operation has been started successfully and the result is returned by way of the callback function with the event `LEVENT_LM_CON_CNF` if the connection is made or `LEVENT_LM_DISCON_IND` if connection fails. The packet is returned by way of the callback with the event `LEVENT_PACKET_HANDLED`.

IR_STATUS_FAILED means the operation failed because of one of the following reasons. Note that the packet is available immediately.

- The connection is busy (already involved in a connection).
- The IrConnect structure is not bound to the stack.
- The packet size exceeds maximum allowed.

IR_STATUS_NO_IRLAP means the operation failed because there is no IrLAP connection (the packet is available immediately).

Comments The result is signaled by way of the callback specified in the IrConnect structure. The callback event LEVENT_LM_CON_CNF indicates that the connection is up and LEVENT_LM_DISCON_IND indicates that the connection failed. Before calling this function the fields in the con structure must be properly set.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrConnectRsp Function

Purpose Accepts an incoming connection that has been signaled by way of the callback with the event LEVENT_LM_CON_IND.

Declared In IrLib.h

Prototype

```
IrStatus IrConnectRsp (UInt16 refNum,
   IrConnect *con, IrPacket *packet,
   UInt8 credit)
```

Parameters

→ *refnum*
IR library refNum.

→ *con*
Pointer to IrConnect structure.

→ *packet*
Pointer to a packet that contains connection data. Even if no connection data is needed, the packet must point to a valid IrPacket structure. The packet is returned by way of the callback with the LEVENT_PACKET_HANDLED event if no errors occur. The maximum size of the packet is IR_MAX_CON_PACKET for an IrLMP connection or IR_MAX_TTP_CON_PACKET for a Tiny TP connection.

→ *credit*
Initial amount of credit advanced to the other side. Must be less than 127. It is ANDed with 0x7f, so if it is greater than 127, unexpected results occur. This parameter is ignored if the connection is an IrLMP connection.

Returns `IR_STATUS_PENDING` means the operation has been started successfully and the packet is returned by way of the callback function with the event `LEVENT_PACKET_HANDLED`.

`IR_STATUS_FAILED` means the operation failed because of one of the following reasons. Note that the packet is available immediately.

- The connection is not in the proper state to require a response.
- The `IrConnect` structure is not bound to the stack.
- The packet size exceeds the maximum allowed.

`IR_STATUS_NO_IRLAP` means the operation failed because there is no IrLAP connection (the packet is available immediately).

Comments `IrConnectRsp` can be called during the callback or later to accept the connection. The type of the connection must already have been set to IrLMP or Tiny TP before the `LEVENT_LM_CON_IND` event.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrDataReq Function

Purpose Sends a data packet.

Declared In `IrLib.h`

Prototype

```
IrStatus IrDataReq (UInt16 refNum,
   IrConnect *con, IrPacket *packet)
```

Parameters

→ *refnum*
IR library refNum.

→ *con*
Pointer to `IrConnect` structure that specifies the connection over which the packet should be sent.

→ *packet*
Pointer to a valid `IrPacket` structure that contains data to send. The packet should not exceed the maximum size found with `IrMaxTxSize()`.

Returns `IR_STATUS_PENDING` means the packet has been queued by the stack. The packet is returned by way of the callback with event `LEVENT_PACKET_HANDLED`.

`IR_STATUS_FAILED` means the operation failed because of one of the following reasons. Note that the packet is available immediately.

- The `IrConnect` structure is not bound to the stack.
- The packet size exceeds the maximum allowed.
- The `IrConnect` structure does not represent an active connection.

Comments The packet is owned by the stack until it is returned by way of the callback with event `LEVENT_PACKET_HANDLED`. The largest packet that can be sent is found by calling `IrMaxTxSize()`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrDisconnectIrLap Function

Purpose Disconnects an IrLAP connection.

Declared In `IrLib.h`

Prototype
```
IrStatus IrDisconnectIrLap (UInt16 refNum)
```

Parameters → *refnum*
IR library refNum.

Returns `IR_STATUS_PENDING` means the operation started successfully and all bound `IrConnect` structures are called back when complete.

`IR_STATUS_NO_IRLAP` means the operation failed because no IrLAP connection exists.

Comments When the IrLAP connection goes down, the callback of all bound `IrConnect` structures is called with event `LEVENT_LAP_DISCON_IND`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrDiscoverReq Function

Purpose Starts an IrLMP discovery process.

Declared In `IrLib.h`

Prototype

```
IrStatus IrDiscoverReq (UInt16 refNum,
   IrConnect *con)
```

Parameters

→ *refnum*
IR library refNum.

→ *con*
Pointer to a bound `IrConnect` structure.

Returns `IR_STATUS_PENDING` means the operation is started successfully; the result is returned by way of callback.

`IR_STATUS_MEDIA_BUSY` means the operation failed because the media is busy. Media busy is caused by one of the following reasons:

- Other devices are using the IR medium.
- A discovery process is already in progress.
- An IrLAP connection exists.

`IR_STATUS_FAILED` means the operation failed because the `IrConnect` structure is not bound to the stack.

Comments The result is signaled by way of the callback function specified in the `IrConnect` structure with the event `LEVENT_DISCOVERY_CNF`. Only one discovery can be invoked at a time.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIsIrLapConnected Function

Purpose Determines if an IrLAP connection exists.

Declared In `IrLib.h`

Prototype

```
BOOL IrIsIrLapConnected (UInt16 refNum)
```

Parameters

→ *refnum*
IR library refNum.

Returns `true` if IrLAP is connected, `false` otherwise.

Comments Only available if `IR_IS_LAP_FUNCS` is defined.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIsMediaBusy Function

Purpose Determines if the IR media is busy.

Declared In `IrLib.h`

Prototype

```
BOOL IrIsMediaBusy (UInt16 refNum)
```

Parameters → *refnum*
IR library refNum.

Returns `true` if IR media is busy, `false` otherwise.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIsNoProgress Function

Purpose Determines if IrLAP is not making progress.

Declared In `IrLib.h`

Prototype

```
BOOL IrIsNoProgress (UInt16 refNum)
```

Parameters → *refnum*
IR library refNum.

Returns `ture` if IrLAP is not making progress, `false` otherwise.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIsRemoteBusy Function

Purpose Determines if IrLAP of the other device is busy.

Declared In `IrLib.h`

Prototype

```
BOOL IrIsRemoteBusy (UInt16 refNum)
```

Parameters → *refnum*
IR library refNum.

Returns `ture` if IrLAP of the other device is busy, `false` otherwise.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrLocalBusy Function

Purpose Sets the IrLAP local busy flag.

Declared In `IrLib.h`

Prototype

```
void IrLocalBusy (UInt16 refNum, BOOL flag)
```

Parameters

→ *refnum*
IR library refNum.

→ *flag*
Value (`true` or `false`) to set for local busy flag of IrLAP.

Returns Returns nothing.

Comments If local busy is set to `true`, then the local IrLAP layer sends RNR (Receive Not Ready) frames to the other side indicating it cannot receive any more data. If the local busy is set to false, IrLAP is ready to receive frames.

The setting takes effect the next time IrLAP sends an RR (Receive Ready) frame. If IrLAP has data to send, the data is sent first, so it should be used carefully.

This function should not be used when using Tiny TP or when multiple connections exist.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrMaxRxSize Function

Purpose Returns the maximum size buffer that can be sent by the other device.

Declared In `IrLib.h`

Prototype

```
UInt16 IrMaxRxSize (UInt16 refNum,
    IrConnect *con)
```

Parameters

→ *refnum*
IR library refNum.

→ *con*
Pointer to `IrConnect` structure that represents an active connection.

Returns Returns the maximum size buffer that can be sent by the other device (maximum bytes that can be received). The value returned is only valid for active connections. The maximum size varies for each connection and is based on the negotiated IrLAP parameters and the type of the connection.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrMaxTxSize Function

Purpose Returns the maximum size allowed for a transmit packet.

Declared In `IrLib.h`

Prototype

```
UInt16 IrMaxTxSize (UInt16 refNum,
    IrConnect *con)
```

Parameters

→ *refnum*
IR library refNum.

→ *con*
Pointer to `IrConnect` structure that represents an active connection.

Returns Returns the maximum size allowed for a transmit packet. The value returned is only valid for active connections. The maximum size varies for each connection and is based on the negotiated IrLAP parameters and the type of the connection.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrOpen Function

Purpose Opens the IR library. This allocates the global memory for the IR stack and reserves the system resources it requires. This must be done before any other IR library calls are made.

Declared In `IrLib.h`

Prototype

```
Err IrOpen (UInt16 refnum, UInt32 options)
```

Parameters

→ *refnum*
IR library refNum. This value is returned from the function `SysLibFind()`, which you must call first to load the IR library.

→ *options*
Open options flags. See the Comments section for details.

Returns Returns 0 if successful.

Comments The following flags can be specified for the `options` parameter to set the speed of the connection:

`irOpenOptSpeed115200`	Set to maximum negotiated baud rate.
`irOpenOptSpeed57600`	Set to 57600 bps (default if no flags given).
`irOpenOptSpeed9600`	Set to 9600 bps.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrSetConTypeLMP Function

Purpose Sets the type of the connection to IrLMP. This function must be called after the `IrConnect` structure is bound to the stack.

Declared In `IrLib.h`

Prototype

```
void IrSetConTypeLMP (IrConnect *con)
```

Parameters

→ *con*
Pointer to `IrConnect` structure.

Returns Returns nothing.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrSetConTypeTTP Function

Purpose Sets the type of the connection to Tiny TP. This function must be called after the `IrConnect` structure is bound to the stack.

Declared In `IrLib.h`

Prototype

```
void IrSetConTypeTTP (IrConnect *con)
```

Parameters

→ *con*
Pointer to `IrConnect` structure.

Returns Returns nothing.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrSetDeviceInfo Function

Purpose Sets the XID info string used during discovery to the given string and length.

Declared In `IrLib.h`

Prototype

```
IrStatus IrSetDeviceInfo (UInt16 refNum,
   UInt8 *info, UInt8 len)
```

Parameters

→ *refnum*
IR library refNum.

→ *info*
Pointer to array of bytes.

→ *len*
Number of bytes pointed to by `info`.

Returns `IR_STATUS_SUCCESS` means the operation is successful.

`IR_STATUS_FAILED` means the operation failed because `info` is too big.

Comments The XID `info` string contains hints and the nickname of the device. The size cannot exceed `IR_MAX_DEVICE_INFO` bytes.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrTestReq Function

Purpose Requests a TEST command frame be sent in the NDM (Normal Disconnect Mode) state.

Declared In `IrLib.h`

Prototype

```
IrStatus IrTestReq (UInt16 refNum,
   IrDeviceAddr devAddr, IrConnect *con,
   IrPacket *packet)
```

Parameters

→ *refnum*
IR library refNum.

→ *devAddr*
Address of device where TEST is sent. This address is not checked so it can be the broadcast address or 0.

→ *con*
Pointer to `IrConnect` structure specifying the callback function to call to report the result.

→ *packet*
Pointer to an `IrPacket` structure that contains the data to send in the TEST command packet. The maximum size data that can be sent is `IR_MAX_TEST_PACKET`. Even if no data is to be sent, a valid packet must be passed.

Returns `IR_STATUS_PENDING` means the operation has been started successfully and the result is returned by way of the callback function with the event `LEVENT_TEST_CNF`. This is also the indication returning the packet.

`IR_STATUS_FAILED` means the operation failed because of one of the following reasons. Note that the packet is available immediately.

- The `IrConnect` structure is not bound to the stack.
- The packet size exceeds the maximum allowed.

`IR_STATUS_MEDIA_BUSY` means the operation failed because the media is busy or the stack is not in the NDM state (the packet is available immediately).

Comments The result is signaled by way of the callback specified in the `IrConnect` structure. The callback event is `LEVENT_TEST_CNF` and the status field indicates the result of the operation. `IR_STATUS_SUCCESS` indicates success and `IR_STATUS_FAILED` indicates no response was received. A packet must be passed

containing the data to send in the TEST frame. The packet is returned when the `LEVENT_TEST_CNF` event is given.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrUnbind Function

Purpose Unbinds the `IrConnect` structure from the protocol stack, freeing its LSAP selector.

Declared In `IrLib.h`

Prototype

```
IrStatus IrUnbind (UInt16 refNum, IrConnect *con)
```

Parameters

→ *refnum*
IR library refNum.

→ *con*
Pointer to `IrConnect` structure to unbind.

Returns `IR_STATUS_SUCCESS` means the operation completed successfully.

`IR_STATUS_FAILED` means the operation failed for one of the following reasons:

- The `IrConnect` structure was not bound.
- The `lLsap` field contained an invalid number.

Compatibility Implemented only if 3.0 New Feature Set is present.

IAS Functions and Macros

This section describes the following functions and macros related to IAS database:

- `IrIAS_Add()`
- `IrIAS_GetInteger()`
- `IrIAS_GetIntLsap()`
- `IrIAS_GetObjectID()`
- `IrIAS_GetOctetString()`
- `IrIAS_GetOctetStringLen()`
- `IrIAS_GetType()`

- IrIAS_GetUserString()
- IrIAS_GetUserStringCharSet()
- IrIAS_GetUserStringLen()
- IrIAS_Next()
- IrIAS_Query()
- IrIAS_SetDeviceName()
- IrIAS_StartResult()

IrIAS_Add Function

Purpose Adds an IAS object to the IAS Database.

Declared In IrLib.h

Prototype

```
IrStatus IrIAS_Add (UInt16 refNum,
    IrIasObject *obj)
```

Parameters

→ *refnum*
IR library refNum.

→ *obj*
Pointer to an IrIASObject structure.

Returns IR_STATUS_SUCCESS means the operation is successful.

IR_STATUS_FAILED means the operation failed for one of the following reasons:

- There is no space in the database.
- An entry with the same class name already exists.
- The attributes of the object violate the IrDA Lite rules (attribute name exceeds IR_MAX_IAS_NAME, or attribute value exceeds IR_MAX_IAS_ATTR_SIZE).
- The class name exceeds IR_MAX_IAS_NAME.

Comments The object is not copied, so the memory for the object must exist for as long as the object is in the database. The IAS database is designed to allow only objects with unique class names, and it checks for this. Class names and attributes names must not exceed IR_MAX_IAS_NAME. Also, attribute values must not exceed IR_MAX_IAS_ATTR_SIZE.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetInteger Macro

Purpose Return an integer value, assuming that the current result item is of type `IAS_ATTRIB_INTEGER`. (Call `IrIAS_GetType()` to determine the type of the current result item.)

Declared In `IrLib.h`

Prototype

```
#define IrIAS_GetInteger (t)
```

Parameters → *t*

Pointer to an `IrIasQuery` structure.

Returns Integer value returned as a `UInt32`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetIntLsap Macro

Purpose Return an integer value that represents an LSAP, assuming that the current result item is of type `IAS_ATTRIB_INTEGER`. (Call `IrIAS_GetType()` to determine the type of the current result item.) Usually integer values returned in a query are LSAP selectors.

Declared In `IrLib.h`

Prototype

```
#define IrIAS_GetIntLsap (t)
```

Parameters → *t*

Pointer to an `IrIasQuery` structure.

Returns Integer value returned as a `UInt8`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetObjectID Macro

Purpose Return the unique object ID of the current result item.

Declared In `IrLib.h`

Prototype
```
#define IrIAS_GetObjectID (t)
```

Parameters → *t*
Pointer to an `IrIasQuery` structure.

Returns Returns the object ID as a `UInt16` type.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetOctetString Macro

Purpose Return a pointer to an octet string, assuming that the current result item is of type IAS_ATTRIB_OCTET_STRING. (Call `IrIAS_GetType()` to determine the type of the current result item.)

Declared In `IrLib.h`

Prototype
```
#define IrIAS_GetOctetString (t)
```

Parameters → *t*
Pointer to an `IrIasQuery` structure.

Returns Pointer to octet string of type `UInt8`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetOctetStringLen Macro

Purpose Return the length of an octet string, assuming that the current result item is of type `IAS_ATTRIB_OCTET_STRING`. (Call `IrIAS_GetType()` to determine the type of the current result item.)

Declared In `IrLib.h`

Prototype
```
#define IrIAS_GetOctetStringLen (t)
```

Parameters → *t*
Pointer to an `IrIasQuery` structure.

Returns Length of octet string returned as a `UInt16`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetType Macro

Purpose Return the type of the current result item.

Declared In `IrLib.h`

Prototype

```
#define IrIAS_GetType (t)
```

Parameters → *t*

Pointer to an `IrIasQuery` structure.

Returns Type of result item, such as `IAS_ATTRIB_INTEGER`, `IAS_ATTRIB_OCTET_STRING` or `IAS_ATTRIB_USER_STRING`. The return value is of type `UInt8`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetUserString Macro

Purpose Return a pointer to a user string, assuming that the current result item is of type `IAS_ATTRIB_USER_STRING`. (Call `IrIAS_GetType()` to determine the type of the current result item.)

Declared In `IrLib.h`

Prototype

```
#define IrIAS_GetUserString (t)
```

Parameters → *t*

Pointer to an `IrIasQuery` structure.

Returns Pointer to result string of type `UInt8`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetUserStringCharSet Macro

Purpose Return the character set of the user string, assuming that the current result item is of type `IAS_ATTRIB_USER_STRING`. (Call

IrIAS_GetType() to determine the type of the current result item.)

Declared In IrLib.h

Prototype
```
#define IrIAS_GetUserStringCharSet (t)
```

Parameters → *t*
Pointer to an IrIasQuery structure.

Returns Character set returned as an IrCharSet value.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_GetUserStringLen Macro

Purpose Return the length of a user string, assuming that the current result item is of type IAS_ATTRIB_USER_STRING. (Call IrIAS_GetType() to determine the type of the current result item.)

Declared In IrLib.h

Prototype
```
#define IrIAS_GetUserStringLen (t)
```

Parameters → *t*
Pointer to an IrIasQuery structure.

Returns Length of user string returned as a UInt8 value.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_Next Function

Purpose Moves the internal pointer to the next result item.

Declared In IrLib.h

Prototype
```
UInt8 *IrIAS_Next (UInt16 refNum,
    IrIasQuery *token)
```

Parameters → *refnum*
IR library refNum.

→ *token*
Pointer to an IrIasQuery structure.

Returns Pointer to the next result item, or 0 if there are no more items.

Comments This function returns a pointer to the start of the next result item. If the pointer is 0, then there are no more result items.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_Query Function

Purpose Makes an IAS query of the IAS database of another device.

Declared In `IrLib.h`

Prototype

```
IrStatus IrIAS_Query (UInt16 refNum,
   IrIasQuery *token)
```

Parameters

→ *refnum*
IR library refNum.

→ *token*
Pointer to an `IrIasQuery` structure initialized as described in the Comments section.

Returns `IR_STATUS_SUCCESS` means the operation is started successfully and the result is signaled by way of the callback function.

`IR_STATUS_FAILED` means the operation failed for one of the following reasons:

- The query exceeds `IR_MAX_QUERY_LEN`.
- The `result` field of `token` is 0.
- The `resultBufSize` field of `token` is 0.
- The `callback` field of `token` is 0.
- A query is already in progress.

`IR_STATUS_NO_IRLAP` means the operation failed because there is no IrLAP connection.

Comments An IrLAP connection must exist to the other device. The IAS query token must be initialized as described below. The result is signaled by calling the callback function whose pointer exists in the `IrIasQuery` structure. Only one query can be made at a time.

The `IrIasQuery` structure passed in the `token` parameter must be initialized as follows:

- Assign a pointer to a callback function in which the result is signaled.
- Set `result` to point to a buffer large enough to hold the result of the query.
- Set `resultBufSize` to the size of the result buffer.
- Set `queryBuf` to point to a valid query.
- Set `queryLen` to the number of bytes in `queryBuf`. The length must not exceed `IR_MAX_QUERY_LEN`.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_SetDeviceName Function

Purpose Sets the `value` field of the device name attribute of the "Device" object in the IAS database.

Declared In `IrLib.h`

Prototype

```
IrStatus IrIAS_SetDeviceName (UInt16 refNum,
   UInt8 *name, UInt8 len)
```

Parameters

→ *`refnum`*
IR library refNum.

→ *`name`*
Pointer to an IAS `value` field for the device name attribute of the device object. It includes the attribute type, character set and device name. This `value` field should be a constant and the pointer must remain valid until `IrIAS_SetDeviceName` is called with another pointer.

→ *`len`*
Total length of the `value` field. Maximum size allowed is `IR_MAX_IAS_ATTR_SIZE`.

Returns `IR_STATUS_SUCCESS` means the operation is successful.

`IR_STATUS_FAILED` means `len` is too big, or the `value` field is not a valid user string.

Compatibility Implemented only if 3.0 New Feature Set is present.

IrIAS_StartResult Macro

Purpose Put the internal pointer to the start of the result buffer.

Declared In `IrLib.h`

Prototype

```
#define IrIAS_StartResult (t)
```

Parameters → *t*

Pointer to an `IrIasQuery` structure.

Returns Returns nothing.

Compatibility Implemented only if 3.0 New Feature Set is present.

Application-Defined Functions

The functions in this section are supplied by the developer and can be named anything.

IrIasQueryCallBack Function

Purpose Signals the result of IAS query. The result of IAS queries is signaled by calling this callback function which is pointed to by the `callBack` field of the `IrIasQuery` structure.

Declared In `IrLib.h`

Prototype

```
void IrIasQueryCallBack (IrStatus status)
```

Parameters → *status*

The status of the query operation. The following values can be passed:

`IR_STATUS_SUCCESS`
: means the query operation finished successfully and the results can be parsed.

`IR_STATUS_DISCONNECT`
: means the link or IrLMP connection was disconnected during the query, so the results are not valid.

Returns Returns nothing.

65

Modem Manager

This chapter provides reference material for the modem manager API. The header file `ModemMgr.h` declares the modem manager API.

Modem Manager Functions

MdmDial Function

Purpose Initialize the modem, dial the phone number and wait for result.

Declared In `ModemMgr.h`

Prototype

```
Err MdmDial (MdmInfoPtr modemP, Char *okDialP,
    Char *userInitP, Char *phoneNumP)
```

Parameters

`modemP`
: Pointer to modem info structure (filled in by caller)

`okDialP`
: (NOT IMPLEMENTED) Pointer to string of chars allowed in dial string

`userInitP`
: Pointer to modem setup string without the AT prefix.

`phoneNumP`
: Pointer to phone number string

Returns 0 if successful; otherwise `mdmErrNoTone`, `mdmErrNoDCD`, `mdmErrBusy`, `mdmErrUserCan`, `mdmErrCmdError`

Comments When executing this function, the system performs these steps:

- Switch to the requested initial baud rate.
- If HW hand-shake is requested, enable CTS/RTS hand-shaking; otherwise, disable it.
- Reset the modem.

- Execute the setup string (if any).
- Configure the modem with required settings.
- Dial the phone number.
- Wait for CONNECT XXXXX or other response.
- If auto-baud is requested, switch to the connected baud rate.

MdmHangUp Function

Purpose Hang up the modem.

Declared In `ModemMgr.h`

Prototype

```
Err MdmHangUp (MdmInfoPtr modemP)
```

Parameters `modemP`
Pointer to modem info structure (filled in by caller)

Returns 0 if successful.

WARNING! This function alters configuration of the serial port (without restoring it).

66

Net Library

This chapter describes the API available in the net library and its Berkeley sockets equivalents. The header file `NetMgr.h` declares the net library API. The chapter covers:

- Net Library Data Structures
- Net Library Constants
- Net Library Functions

For more information on the net library, see the chapter "Network Communication" in the *Palm OS Programmer's Companion*, vol. II, *Communications*.

Net Library Data Structures

NetConfigNameType Struct

Purpose The `NetConfigNameType` structure defines a configuration name. A configuration is a specific set of values for the net library settings. Typically, users set up configurations and assign names to them using the Network preferences panel.

Prototype

```
typedef struct {
  Char name[netConfigNameSize];
} NetConfigNameType, NetConfigNamePtr;
```

`name` is the configuration's name. The `netConfigNameSize` constant is currently defined to be 32.

Compatibility Supported only if 3.2 New Feature Set is present.

NetHostInfoBufType Struct

Purpose The `NetHostInfoBufType` struct contains information about a host. The `NetHostInfoType` struct, which maps to the `hostent` struct, points to fields in this struct for its information.

Prototype

```
typedef struct {
  NetHostInfoType  hostInfo;
  Char name[netDNSMaxDomainName+1];
  Char *aliasList[netDNSMaxAliases+1];
  Char aliases[netDNSMaxAliases]
[netDNSMaxAliases+1];
  NetIPAddr *addressList[netDNSMaxAddresses];
  NetIPAddr address[netDNSMaxAddresses];
} NetHostInfoBufType, *NetHostInfoBufPtr;
```

Fields

`hostInfo`
: A `NetHostInfoType` struct, which maps to the Berkeley UNIX sockets `hostent` structure.

`name`
: Official host name.

`aliasList`

`aliases`
: An array of aliases for the host name.

`addressList`

`address`
: An array of pointers to 32-bit IP addresses in host byte order.

NetHostInfoType Struct

Purpose The `NetHostInfoType` structure maps to the Berkeley UNIX sockets `hostent` structure. It is defined as follows:

Prototype

```
typedef struct {
  Char *nameP;
  Char **nameAliasesP;
  UInt16 addrType;
  UInt16 addrLen;
  UInt8 **addrListP;
} NetHostInfoType, *NetHostInfoPtr;
```

Fields

`nameP`
Official host name.

`nameAliasesP`
An array of aliases for the host name.

`addrType`
The type of the return addresses. See `NetSocketAddrEnum`.

`addrLen`
The length in bytes of the return addresses.

`addrListP`
An array of pointers to addresses in host byte order.

NetServInfoBufType Struct

Purpose

The `NetServInfoBufType` struct contains information about a service. The `NetServInfoType` struct, which maps to the `servent` struct, points to fields in this struct for much of its information.

Prototype

```
struct {
  NetServInfoType servInfo;
  Char name[netServMaxName+1];
  Char *aliasList[netServMaxAliases+1];
  Char aliases[netServMaxAliases]
[netServMaxName];
  Char protoName[netProtoMaxName+1];
  UInt8 reserved;
} NetServInfoBufType, *NetServInfoBufPtr;
```

Fields

`servInfo`
A `NetServInfoType` struct, which maps to the Berkeley UNIX sockets `servent` structure.

`name`
Official name of the service

`aliasList`

`aliases`
Array of aliases for the service name.

`protoName`
Name of the protocol to use.

`reserved`
Reserved for future use.

NetServInfoType Struct

Purpose The `NetServInfoType` structure maps to the `servent` structure in Berkeley UNIX sockets API. It contains information about a service.

Prototype

```
struct {
  Char *nameP;
  Char **nameAliasesP;
  UInt16 port;
  Char *protoP;
} NetServInfoType, *NetServInfoPtr;
```

Fields

`nameP`
Official name of the service

`nameAliasesP`
Array of aliases for the service name.

`port`
Port number for the service.

`protoP`
Name of the protocol to use.

NetSocketAddrEnum Enum

Purpose The `NetSocketAddrEnum` enum specifies the address types supported by the net library.

Prototype

```
typedef enum {
  netSocketAddrRaw = 0,
  netSocketAddrINET = 2
} NetSocketAddrEnum
```

Fields

`netSocketAddrRaw`
Raw address. Supported in Palm OS® version 3.0 and higher.

`netSocketAddrINET`
IP address.

NetSocketAddrINType Struct

Purpose The `NetSocketAddrINType` struct holds an internet socket address, that is, a socket that uses one of the internet protocols. This structure directly maps to the BSD UNIX `sockaddr_in` structure.

Prototype

```
typedef struct NetSocketAddrINType {
  Int16 family;
  UInt16  port;
  NetIPAddr addr;
} NetSocketAddrINType;
```

Fields

`family`
: Address family in host byte order. This is either `netSocketAddrINET` or `netSocketAddrRaw`.

`port`
: The port in network byte order.

`addr`
: The IP address in network byte order.

NetSocketAddrRawType Struct

Purpose The `NetSocketAddrRawType` structure holds a raw socket address.

Prototype

```
typedef struct NetSocketAddrRawType {
  Int16 family;
  UInt16 ifInstance;
  UInt32 ifCreator;
} NetSocketAddrRawType;
```

Fields

`family`
: Address family in host byte order. This is either `netSocketAddrINET` or `netSocketAddrRaw`.

`ifInstance`
: The instance number of the interface that the socket uses to send and receive data.

`ifCreator`
: The creator of the interface that the socket uses.

Compatibility Raw sockets are supported in Palm OS version 3.0 and higher.

NetSocketAddrType Struct

Purpose The `NetSocketAddrType` structure holds a generic socket address. This struct can hold any type of address including Internet addresses. It directly maps to the BSD UNIX `sockaddr` structure.

Note that this structure is the same size as `NetSocketAddrINType` and `NetSocketAddrRawType`. This means that one of those two structures can be used for parameters declared to be `NetSocketAddrType`.

Prototype

```
typedef struct NetSocketAddrType {
  Int16 family;
  UInt8 data[14];
} NetSocketAddrType;
```

NetSocketNoticeType Struct

Purpose The `NetSocketNoticeType` structure specifies the socket state changes that your application wishes to register for and the means that Net Library should use to communicate with your application—for example, by notification.

When registering your application for a socket notice, call `NetLibSocketOptionSet()` and pass in a pointer to a `NetSocketNoticeType` structure.

Prototype

```
typedef struct NetSocketNoticeType {
  Int32 condition;
  NoticeTypeEnum type;
  union{
    struct {
      UInt32 notifyType;
    }notify
    ...
  } notice;
} NetSocketNoticeType;
```

Fields

`condition`
: Bitifield specifying what socket activity you wish to receive notices for—closing a socket, socket receiving TCP data, etc. Constants for the bits in this field are defined in `NetMgr.h`. See "Socket Notice Trigger Conditions" on page 1468.

`type`
: How net library should communicate with your application. For instance, `netSocketNoticeNotify` indicates a socket notice that uses notifications (This is the only socket notice type supported in Palm OS Garnet version 5.4). Socket notice type values are defined in `NoticeTypeEnum`.

`notice`
: Union that contains a structure describing the socket notice. The structure is appropriate to the kind of socket notice, as specified in the `type` field.

`notify`
: Structure describing the notification that the socket notice is sending to your application.

`notifyType`
: Socket notification constant that you defined for your application. It must be unique.

Compatibility Implemented if 5.4 New Feature Set is present.

NetSocketRef Typedef

Purpose The `NetSocketRef` defines a socket descriptor. The socket descriptor is created and returned by `NetLibSocketOpen()`. It is used in any function that requires access to a socket.

Prototype

```
typedef Int16 NetSocketRef
```

NetSocketTypeEnum Enum

Purpose The `NetSocketTypeEnum` enum specifies the available socket types.

Prototype

```
typedef enum {
  netSocketTypeStream=1,
  netSocketTypeDatagram=2,
  netSocketTypeRaw=3,
  netSocketTypeReliableMsg=4
} NetSocketTypeEnum
```

Constants

`netSocketTypeStream`
: Streams protocol over wireline.

`netSocketTypeDatagram`
: UDP protocol.

`netSocketTypeRaw`
: Raw mode.

Compatibility Implemented if 5.4 New Feature Set is present.

NoticeTypeEnum Enum

Purpose The `NoticeTypeEnum` specifies the kind of socket notice that an application is registering for. It indicates by what means a change in socket state should be communicated to the application—for instance, by notifications. Use a `NoticeTypeEnum` value for the `type` field of `NetSocketNoticeType`.

Prototype

```
typedef enum {
  netSocketNoticeNotify=1,
  netSocketNoticeEvent
  netSocketNoticeMailbox
  netSocketNoticeCallback
  netSocketNoticeWake
} NoticeTypeEnum;
```

Constants

`netSocketNoticeNotify`
: Indicates that socket notice should use notifications to communicate with the application.

`netSocketNoticeEvent`
: Not supported in Palm OS Garnet version 5.4.

`netSocketNoticeMailbox`
: Not supported in Palm OS Garnet version 5.4.

`netSocketNoticeCallback`
: Not supported in Palm OS Garnet version 5.4.

`netSocketNoticeWake`
: Not supported in Palm OS Garnet version 5.4.

Compatibility Implemented if 5.4 New Feature Set is present.

Net Library Constants

Configuration Aliases

A configuration is a set of specific values for the net library settings. The net library defines a set of built-in configuration aliases for common network setups. These aliases point to configurations instead of holding the actual values themselves. You can specify an alias anywhere in the API you would specify a configuration.

The constants listed here specify the alias names. Most of the net library API requires a configuration index rather than a name. Use `NetLibConfigIndexFromName()` to obtain the alias's index from the name.

Constants

`netCfgNameDefault`
: The default configuration.

`netCfgNameDefWireline`
: The default configuration for wireline communications.

`netCfgNameDefWireless`
: The default configuration for wireless communications.

`netCfgNameCTPWireline`
: The default configuration for wireline communications through the Palm Web Clipping Proxy server.

`netCfgNameCTPWireless`
: The default configuration for wireless communications through the Palm Web Clipping Proxy server.

Comments

By default, `netCfgNameDefault` points to the user's default configuration, and all other aliases point to `netCfgNameDefault` except for `netCfgNameCTPWireless`, which points to an private wireless configuration.

Compatibility

Supported on version 3.2 and later.

I/O Flags

The I/O flags specify special handling instructions to functions that send and receive data. You can OR these values together to specify more than one.

Constants

`netIOFlagOutOfBand`
: Process out-of-band data. Available for send calls only.

`netIOFlagPeek`
: Peek at incoming message without dequeuing it.

`netIOFlagDontRoute`
: Send without using routing. This constant is currently ignored.

Socket Notice Trigger Conditions

The socket notice trigger condition is a bitfield that identifies one or more of the state changes associated with a socket—for instance, closing a socket, a socket receiving TCP data, etc.

When you register to receive a socket notice, the bitfield is passed to `NetLibSocketOptionSet()` in the `condition` field of the `NetSocketNoticeType` structure. When you receive a socket notification, the `condition` field of the `SysNotifyNetSocketType` structure contains this bitfield, which you read to determine what state changes the socket has undergone.

These constants are defined in `NetMgr.h` as follows:

Constants

`netSocketNoticeErr`
: Socket error.

`netSocketNoticeUDPReceive`
: Socket received UDP data.

`netSocketNoticeTCPReceive`
: Socket received TCP data.

`netSocketNoticeTCPTransmit`
: Socket transmitted TCP data.

`netSocketNoticeTCPRemoteClosed`
: Remote TCP connection closed.

`netSocketNoticeTCPClosed`
Local TCP connection closed.

`netSocketNoticeConnectInbound`
A remote node tried to connect.

`netSocketNoticeConnectOutbound`
Your application tried to connect to a remote node.

Comments f you wish to receive notice of *all* socket conditions, use `0xFFFFFFFF` as the value of the *`condition`* field .

Compatibility Implemented if 5.4 New Feature Set is present.

Tracing Bits

The tracing bits are used to set the level of event tracing. An application can get a list of events in the trace buffer using the `NetLibMaster()` call.

You can set the tracing for each network interface using `NetLibIFSettingSet()` and for the net library in general with `NetLibSettingSet()`.

Constants

`netTracingErrors`
Record run-time errors. This is the default.

`netTracingMsgs`
Record application trace messages.

`netTracingPkts`
Record packet I/O. This bit is obsolete in versions 3.2 and higher, but is mapped to `netTracingPktIP`.

`netTracingFuncs`
Record function flow.

`netTracingAppMsgs`
Record application messages sent using `NetLibTracePrintF()` and `NetLibTracePutS()`.

`netTracingPktIP`
Record packet I/O. If this set, the following five options are enabled.

`netTracingData40`
: Record the first 40 bytes of each packet sent or received. This option is mutually exclusive with `netTracingData`.

`netTracingData`
: Record the entirety of each packet sent or received. This option is mutually exclusive with `netTracingData40`.

`netTracingIFHi`
: Record packets sent or received at the highest layer of the network interface. This layer is just below the IP layer.

`netTracingIFMid`
: Record packets sent or received at the layer just below the highest layer of the network interface.

`netTracingIFLow`
: Record packets sent or received at the lowest layer of the network interface.

Compatibility The `netTracingPktXXX` constants are supported only in version 3.2 devices and higher. In previous versions, specify `netTracingPkts` instead; only the size of the packet is recorded.

Net Library Functions

NetHToNL Macro

Purpose Converts a 32-bit value from host to network byte order.

Declared In `NetBitUtils.h`

Prototype
```
#define NetHToNL (x)
```

Parameters → *x*
: 32-bit value to convert.

Returns Returns `x` in network byte order.

Sockets Equivalent `htonl()`

See Also `NetNToHS()`, `NetNToHL()`, `NetHToNS()`

NetHToNS Macro

Purpose Converts a 16-bit value from host to network byte order.

Declared In `NetBitUtils.h`

Prototype
```
#define NetHToNS (x)
```

Parameters → *x*
16-bit value to convert.

Returns Returns `x` in network byte order.

Sockets Equivalent `htons()`

See Also `NetNToHS()`, `NetNToHL()`, `NetHToNL()`

NetLibAddrAToIN Function

Purpose Converts an ASCII string representing a dotted decimal IP address into a 32-bit IP address in network byte order.

Declared In `NetMgr.h`

Prototype
```
NetIPAddr NetLibAddrAToIN (UInt16 libRefnum,
    const Char *a)
```

Parameters → *libRefNum*
Reference number of the net library.

→ *a*
Pointer to ASCII dotted decimal string.

Returns Returns a 32-bit network byte order IP address or -1 if `a` doesn't represent a dotted decimal IP address

Sockets Equivalent `UInt32 inet_addr (char *cp)`

See Also `NetLibAddrINToA()`

NetLibAddrINToA Function

Purpose Converts an IP address from 32-bit network byte order into a dotted decimal ASCII string.

Declared In `NetMgr.h`

Prototype

```
Char *NetLibAddrINToA (UInt16 libRefnum,
   NetIPAddr inet, Char *spaceP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *inet*
32-bit IP address in network byte order.

← *spaceP*
Buffer used to hold the return value.

Returns Returns in `spaceP` the dotted decimal ASCII string representation of the IP address.

Sockets Equivalent

```
char *inet_ntoa (struct in_addr in)
```

See Also `NetLibAddrAToIN()`

NetLibClose Function

Purpose Closes the net library.

Declared In `NetMgr.h`

Prototype

```
Err NetLibClose (UInt16 libRefnum,
   UInt16 immediate)
```

Parameters

→ *libRefnum*
Reference number of the net library.

→ *immediate*
If `true`, library will shut down immediately. If `false`, library will shut down only if close timer expires before another `NetLibOpen()` is issued.

Returns Returns one of the following values:

0
Success.

`netErrNotOpen`
: Library was not open.

`netErrStillOpen`
: Not really an error; returned if library is still in use by another application.

Sockets Equivalent

None.

Comments

Applications must call this function when they no longer need the net library. If the net library open count is greater than 1 before this call is made, the count is decremented and `netErrStillOpen` is returned. If the open count was 1, the library takes the following action:

- If `immediate` is `true`, the library shuts down immediately. All network interfaces are brought down, the net protocol stack task is terminated, and all memory used by the net library is freed.
- If `immediate` is `false`, a close timer is created and this call returns immediately without actually bringing the net library down. Instead it leaves it up and running but marks it as in the "close-wait" state. It remains in this state until either the timer expires or another `NetLibOpen` is issued. If the timer expires, the library is shut down. If another `NetLibOpen` call is issued before the timer expires (possibly by another application), the timer is cancelled and the library is marked as fully open.

In most cases, you should pass `false` for `immediate`. This allows the user to quit one Internet application and launch another within a short period of time without having to wait through the process of closing down and then re-establishing dial-up network connections.

See Also

`NetLibOpen()`, `NetLibOpenCount()`

NetLibConfigAliasGet Function

Purpose Return the configuration that an alias points to.

Prototype

```
Err NetLibConfigAliasGet (UInt16 refNum,
   UInt16 aliasIndex, UInt16 *indexP,
   Boolean *isAnotherAliasP)
```

Parameters

→ *refNum*
Reference number of the net library.

→ *aliasIndex*
Index of the alias.

← *indexP*
Index of the configuration pointed to by the alias.

← *isAnotherAliasP*
`true` if `indexP` is the index of another alias; `false` if `indexP` specifies an actual configuration.

Returns Returns one of the following values:

`0`
Success.

`netErrConfigNotAlias`
The configuration at `aliasIndex` is not an alias.

`netErrOutOfCmdBlocks`

`netErrParamErr`
The specified index is out of range or there is no configuration at the index.

Sockets Equivalent None

Comments Use this routine to find out which configuration a built-in alias points to. See "Configuration Aliases" for a description of the built-in aliases.

Compatibility Implemented only if 3.2 New Feature Set is present.

See Also `NetLibConfigAliasSet()`

NetLibConfigAliasSet Function

Purpose

Set a built-in alias to point to a defined configuration.

Prototype

```
Err NetLibConfigAliasSet (UInt16 refNum,
   UInt16 configIndex, UInt16 aliasToIndex)
```

Parameters

→ *refNum*
Reference number of the net library.

→ *configIndex*
Index of the built-in alias to be set.

→ *aliasToIndex*
Index of the configuration to which the alias should point. You cannot set an alias to point to itself.

Returns

Returns one of the following values:

`0`
Success.

`netErrConfigCantPointToAlias`
The configuration at `aliasToIndex` is an alias that points to an alias.

`netErrConfigNotAlias`
The configuration at `configIndex` isn't an alias.

`netErrOutOfCmdBlocks`

`netErrParamErr`
The specified index is out of range or there's no configuration at the index.

Sockets Equivalent

None

Comments

This function is used by the Network preferences panel when the user edits a configuration. Your application can use it to associate any of the built-in aliases with a defined configuration.

The built-in aliases are typically set up as shown in Table 66.1. In this example, applications that specify a configuration index of 0 through 3 use a configuration that the user defines. Applications that use index 4 use a private configuration created by the network library.

Table 66.1 Example Configuration Table

Index	Name	Alias To
0	.Default	6
1	.DefWireline	0
2	.DefWireless	0
3	.CTPWireline	0
4	.CTPWireless	5
5	_RAMCTP	
6	user-defined	

An alias can point to another alias so long as the nesting level is only one deep. That is, if you point an alias to an alias, you'll receive an error if that alias in turn points to another alias. This eliminates the possibility that an alias never resolves to an actual configuration.

Compatibility Implemented only if 3.2 New Feature Set is present.

See Also `NetLibConfigAliasGet()`

NetLibConfigDelete Function

Purpose Delete a configuration from the net library's configuration table.

Prototype

```
Err NetLibConfigDelete (UInt16 refNum,
   UInt16 index)
```

Parameters

→ *refNum*
Reference number of the net library.

→ *index*
Index of the configuration to delete. You cannot delete one of the built-in aliases described in "`Configuration Aliases`."

Returns Returns one of the following values:

0
Success.

`netErrConfigCantDelete`
The configuration at `index` is a built-in alias.

`netErrOutOfCmdBlocks`

`netErrParamErr`
The specified index is out of range.

Sockets Equivalent None

Compatibility Implemented only if 3.2 New Feature Set Set is present.

See Also `NetLibConfigSaveAs()`

NetLibConfigIndexFromName Function

Purpose Obtain a configuration's index given its name.

Prototype

```
Err NetLibConfigIndexFromName (UInt16 refNum,
   NetConfigNamePtr nameP, UInt16 *indexP)
```

Parameters

→ *refNum*
Reference number of the net library.

→ *nameP*
Pointer to a configuration name. See `NetConfigNameType`.

← *indexP*
The index of the configuration with the name `*nameP`.

Returns Returns one of the following values:

`0`
Success.

`netErrConfigNotFound`
A configuration with the specified name could not be found.

`netErrOutOfCmdBlocks`

Sockets Equivalent None

Comments This function returns the index of a configuration given its name. Your application should store the configuration's index rather than its name because a configuration's name can change.

If you pass the name of a built-in alias in `nameP`, this function returns the index of the alias's entry in the configuration table; it does not return the index that the alias points to. For example, if the alias `netCfgNameCTPWireless` is stored at index 4 and points to index 5, `NetLibConfigIndexFromName` returns 4. If you want to obtain the index that an alias points to, use `NetLibConfigAliasGet()`.

Compatibility Implemented only if 3.2 New Feature Set is present.

See Also `NetLibConfigList()`

NetLibConfigList Function

Purpose Return a list of net library configuration names.

Prototype

```
Err NetLibConfigList (UInt16 refNum,
   NetConfigNameType nameArray[],
   UInt16 *arrayEntriesP)
```

Parameters

→ *refNum*
: Reference number of the net library.

← *nameArray*
: The list of defined configurations. See `NetConfigNameType`.

↔ *arrayEntriesP*
: On entry, contains the number of elements in `nameArray`. On return, contains the number of elements in `nameArray` that were actually used. The Net Library currently returns up to 16 entries. If the array is not large enough to hold all the configuration names, this function returns only as many names as the array can hold.

Returns Returns one of the following values:

0
: Success.

`netErrOutOfCmdBlocks`

Sockets Equivalent None

Comments Use this function to obtain a list of the names of defined network configurations and configuration aliases.

Users create specific configurations using the Network preferences panel and associate names with each configuration. This function returns the list of defined configurations.

In addition to user-defined configurations, this function also returns built-in configuration aliases and private configurations. The built-in configuration aliases are described in "`Configuration Aliases`." Their actual names begin with a period (.). Private configurations have names that begin with an underscore (_).

IMPORTANT: If you present the list returned by this function to your application's users, you must first filter out names beginning with a period or an underscore. These names are for internal use only.

Your application should refer to a configuration by its index rather than its name because the name can be changed. To obtain the configuration's index from its name, use `NetLibConfigIndexFromName()`.

Compatibility Implemented only if 3.2 New Feature Set is present.

NetLibConfigMakeActive Function

Purpose Makes the specified configuration current without opening the net library.

Prototype

```
Err NetLibConfigMakeActive (UInt16 refNum,
   UInt16 configIndex)
```

Parameters

→ *refNum*
Reference number of the net library.

→ *configIndex*
Index of the configuration to use. An index of 0 refers to the default configuration as defined by the Network preferences panel.

Returns Returns one of the following values:

`0`
Success.

`netErrBufTooSmall`

`netErrConfigAliasErr`

`netErrConfigCantDelete`

`netErrConfigEmpty`

`netErrConfigNotFound`

`netErrOutOfCmdBlocks`

`netErrParamErr`

`netErrPrefNotFound`

Sockets Equivalent None

Comments This function is used mainly by the Network preferences panel when the user edits and saves network configurations. The Network preferences panel uses this function to make current the configuration the user wants to edit, set the settings appropriately, and then save the configuration using `NetLibConfigSaveAs()`.

Use this routine to make a specific configuration the current configuration without opening the net library. You should not use it if the net library is already open.

Unlike `NetLibOpenConfig()`, this routine does not save the current net library configuration so that it can be restored upon close.

Compatibility Implemented only if 3.2 New Feature Set is present.

NetLibConfigRename Function

Purpose Rename the specified configuration.

Prototype

```
Err NetLibConfigRename (UInt16 refNum,
    UInt16 index, NetConfigNamePtr newNameP)
```

Parameters

→ *refNum*
Reference number of the net library.

→ *configIndex*
Index of the configuration to be renamed.

→ *newNameP*
Pointer to the new name. See `NetConfigNameType`. The new name must not start with a period (.) or an underscore (_).

Returns Returns one of the following values:

`0`
Success.

`netErrConfigBadName`
The new name begins with a period.

`netErrConfigCantDelete`
The configuration at the specified index is a built-in alias or private configuration that cannot be renamed.

`netErrOutOfCmdBlocks`

`netErrParamErr`
The specified index is out of range or there is no configuration at the index.

Sockets Equivalent None

Comments You cannot specify a name beginning with a period (.) or an underscore (_). Names beginning with a period are reserved for the built-in configuration aliases. Names beginning with an underscore are hidden configurations used internally by net library.

Compatibility Implemented only if 3.2 New Feature Set is present.

NetLibConfigSaveAs Function

Purpose Save the current net library settings as a configuration with the specified name.

Prototype
```
Err NetLibConfigSaveAs (UInt16 refNum,
   NetConfigNamePtr nameP)
```

Parameters → *refNum*
Reference number of the net library.

→ *nameP*
: Pointer to a name for the configuration. See `NetConfigNameType`. The name must not start with a period (.) or an underscore (_).

Returns Returns one of the following values:

`0`
: Success.

`netErrConfigBadName`
: The specified name begins with a period or underscore.

`netErrConfigTooMany`
: Not enough space to add another configuration. The Net Library can hold up to 16 configuration.

`netErrOutOfCmdBlocks`

Sockets Equivalent None

Comments If the name you specify already exists, its configuration is replaced with this configuration.

You cannot specify a name beginning with a period (.) or an underscore (_). Names beginning with a period are reserved for the built-in configuration aliases. Names beginning with an underscore are hidden configurations used internally by net library.

The net library assigns an index to this new configuration. The configuration's index remains constant, while its name may change. Use `NetLibConfigIndexFromName()` to obtain the configuration's index.

Compatibility Implemented only if 3.2 New Feature Set is present.

See Also `NetLibConfigDelete()`, `NetLibConfigRename()`

NetLibConnectionRefresh Function

Purpose

This routine is a convenience call for applications. It checks the status of all connections and optionally tries to open any that were closed.

Declared In

`NetMgr.h`

Prototype

```
Err NetLibConnectionRefresh (UInt16 refNum,
   Boolean refresh, UInt8 *allInterfacesUpP,
   UInt16 *netIFErrP)
```

Parameters

→ *refnum*
: Reference number of the net library.

→ *refresh*
: If `true`, any connections that aren't currently open are opened.

← *allInterfacesUpP*
: Set to `true` if all connections are open.

← *netIFErrP*
: First error encountered when reopening connections that were closed. (See `NetLibIFUp()` for a list of possible values.)

Returns

`Returns one of the following values:`

`0`
: Success.

`netErrBufTooSmall`

`netErrOutOfCmdBlocks`

`netErrNoInterfaces`

Sockets Equivalent

None.

Comments

This function determines whether a connection is up based on the internal status of the TCP/IP stack. To test the presence of a "physical connection" (phone line, modem, serial cable), a command should be sent once it's been determined that the logical connection is up. If the physical connection is broken, nothing returns and a timeout error eventually occurs.

NetLibDmReceive Function

Purpose Receive data from a socket directly into a database record.

Declared In `NetMgr.h`

Prototype

```
Int16 NetLibDmReceive (UInt16 libRefNum,
   NetSocketRef socket, void *recordP,
   UInt32 recordOffset, UInt16 rcvLen,
   UInt16 flags,
   void *fromAddrP, UInt16 *fromLenP,
   Int32 timeout,
   Err *errP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *socket*
Descriptor for the open socket.

← *recordP*
Pointer to beginning of record to receive data into. Must be locked for use.

→ *recordOffset*
Offset from beginning of record to read data into.

→ *rcvLen*
Maximum number of bytes to read.

→ *flags*
One or more `netIOFlagxxx` flags. See "I/O Flags."

← *fromAddrP*
Pointer to buffer to hold address of sender (a `NetSocketAddrType` struct). Pass `NULL` if you don't need sender information.

↔ *fromLenP*
On entry, size of `fromAddrP` buffer. On exit, actual size of returned address in `fromAddrP`. Pass `NULL` if you don't need sender information.

→ *timeout*
Maximum timeout in system ticks; -1 means wait forever.

← *errP*
Contains an error code if the return value is -1.

Returns Returns the number of bytes successfully received. If the return value is 0, the socket has been shut down by the remote host. If the return value is -1, an error has occurred and `errP` contains one of the following values:

`0`
: No error.

`netErrTimeout`
: Call timed out.

`netErrNotOpen`
: The referenced net library has not been opened yet.

`netErrParamErr`

`netErrSocketNotOpen`

`netErrWouldBlock`

`netErrUserCancel`

`netErrOutOfMemory`

Comments This call behaves similarly to `NetLibReceive()` but reads the data directly into a database record, which is normally write-protected. The caller must pass a pointer to the start of the record and an offset into the record of where to start the read.

NetLibFinishCloseWait Function

Purpose Forces the net library to do a complete close if it's currently in the close-wait state.

Declared In `NetMgr.h`

Prototype

```
Err NetLibFinishCloseWait (UInt16 libRefnum)
```

Parameters → *`libRefnum`*
: Reference number of the net library.

Returns `Returns one of the following values:`

`0`
: Success.

`netErrTimeout`

Sockets Equivalent None.

Comments This call checks the current open state of the net library. If it's in the close-wait state (see `NetLibClose()`), it forces the library to perform an immediate, complete close operation.

NetLibGetHostByAddr Function

Purpose Looks up a host name given its IP address.

Declared In `NetMgr.h`

Prototype

```
NetHostInfoPtr NetLibGetHostByAddr
   (UInt16 libRefNum, UInt8 *addrP, UInt16 len,
   UInt16 type, NetHostInfoBufPtr bufP,
   Int32 timeout, Err *errP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *addrP*
IP address of host to lookup.

→ *len*
Length, in bytes, of `*addrP`.

→ *type*
Type of `addrP`. See `NetSocketAddrEnum`.

← *bufP*
Pointer to a `NetHostInfoBufType` struct in which to store the results of the lookup.

→ *timeout*
Maximum timeout in system ticks; -1 means wait forever.

← *errP*
Contains an error code if the return value is 0.

Returns Returns a pointer to the `NetHostInfoType` portion of `bufP` that contains results of the lookup. If the return value is 0, an error has occurred, and `errP` contains one of the following values:

0
No error

`netErrTimeout`
: Call timed out.

`netErrNotOpen`
: The referenced net library has not been opened yet.

`netErrDNSNameTooLong`

`netErrDNSBadName`

`netErrDNSLabelTooLong`

`netErrDNSAllocationFailure`

`netErrDNSTimeout`

`netErrDNSUnreachable`

`netErrDNSFormat`

`netErrDNSServerFailure`

`netErrDNSNonexistantName`

`netErrDNSNIY`

`netErrDNSRefused`

`netErrDNSImpossible`

`netErrDNSNoRRS`

`netErrDNSAborted`

`netErrDNSBadProtocol`

`netErrDNSTruncated`

`netErrDNSNoRecursion`

`netErrDNSIrrelevant`

`netErrDNSNotInLocalCache`

`netErrDNSNoPort`

Sockets Equivalent

```
struct hostent *gethostbyaddr (char *addr,
int len, int type);
```

Comments

This call queries the domain name server(s) to look up a host name given its IP address.

See Also

`NetLibGetHostByName()`

NetLibGetHostByName Function

Purpose Looks up a host IP address given a host name.

Declared In `NetMgr.h`

Prototype

```
NetHostInfoPtr NetLibGetHostByName
   (UInt16 libRefNum, const Char *nameP,
   NetHostInfoBufPtr bufP, Int32 timeout,
   Err *errP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *nameP*
Name of host to look up.

← *bufP*
Pointer to a `NetHostInfoBufType` struct in which to store the results of the lookup.

→ *timeout*
Maximum timeout in system ticks; -1 means wait forever.

← *errP*
Contains an error code if the return value is 0.

Returns Returns a pointer to the `NetHostInfoType` portion of `bufP`, which contains results of the lookup. If the return value is 0, an error has occurred and `errP` contains one of the following values:

`0`
No error

`netErrTimeout`
Call timed out.

`netErrNotOpen`
The referenced net library has not been opened yet.

`netErrDNSNameTooLong`

`netErrDNSBadName`

`netErrDNSLabelTooLong`

`netErrDNSAllocationFailure`

`netErrDNSTimeout`

`netErrDNSUnreachable`

```
netErrDNSFormat
netErrDNSServerFailure
netErrDNSNonexistantName
netErrDNSNIY
netErrDNSRefused
netErrDNSImpossible
netErrDNSNoRRS
netErrDNSAborted
netErrDNSBadProtocol
netErrDNSTruncated
netErrDNSNoRecursion
netErrDNSIrrelevant
netErrDNSNotInLocalCache
netErrDNSNoPort
```

Sockets Equivalent

```
struct hostent *gethostbyname (char *name);
```

Comments

→ *This call first checks the local name IP address host table in the net library preferences. If the entry is not found, it then queries the domain name server(s).*

See Also

NetLibGetHostByAddr(), NetLibGetMailExchangeByName()

NetLibGetMailExchangeByName Function

Purpose Looks up the name of a host to use for a given mail exchange.

Declared In `NetMgr.h`

Prototype

```
Int16 NetLibGetMailExchangeByName
   (UInt16 libRefNum, Char *mailNameP,
   UInt16 maxEntries, Char hostNames[][255+1],
   UInt16 priorities[], Int32 timeout, Err *errP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *mailNameP*
Name of the mail exchange to look up.

→ *maxEntries*
Maximum number of host names to return.

← *hostNames*
Array of character strings of length `255+1`. The host name results are stored in this array. This array must be able to hold at least `maxEntries` host names.

← *priorities*
Array of `Word`s. The priorities of each host name found are stored in this array. This array must be at least `maxEntries` in length.

→ *timeout*
Maximum timeout in system ticks; -1 means wait forever.

← *errP*
Contains an error code if the return value is less than 0.

Returns Returns the number of entries successfully found. If the return value is a negative number, an error has occurred, and `errP` contains one of the following values:

`0`
No error

`netErrTimeout`
Call timed out.

`netErrNotOpen`
The referenced net library has not been opened yet.

`netErrDNSNameTooLong`

`netErrDNSBadName`

`netErrDNSLabelTooLong`

`netErrDNSAllocationFailure`

`netErrDNSTimeout`

`netErrDNSUnreachable`

`netErrDNSFormat`

`netErrDNSServerFailure`

`netErrDNSNonexistantName`

`netErrDNSNIY`

`netErrDNSRefused`

`netErrDNSImpossible`

`netErrDNSNoRRS`

`netErrDNSAborted`

`netErrDNSBadProtocol`

`netErrDNSTruncated`

`netErrDNSNoRecursion`

`netErrDNSIrrelevant`

`netErrDNSNotInLocalCache`

`netErrDNSNoPort`

Sockets Equivalent None

Comments This call looks up the name(s) of host(s) to use for sending an e-mail. The caller passes the name of the mail exchange in `mailNameP` and gets back a list of host names to which the mail message can be sent.

See Also `NetLibGetHostByAddr()`, `NetLibGetHostByName()`

NetLibGetServByName Function

Purpose Looks up the port number for a standard TCP/IP service, given the desired protocol.

Declared In `NetMgr.h`

Prototype

```
NetServInfoPtr NetLibGetServByName
    (UInt16 libRefNum, const Char *servNameP,
    const Char *protoNameP,
    NetServInfoBufPtr bufP,
    Int32 timeout, Err *errP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *servNameP*
Name of the service to look up. Possible services are "echo", "discard", "daytime", "qotd", "chargen", "ftp-data", "ftp", "telnet", "smtp", "time", "name", "finger", "pop2", "pop3", "nntp", "imap2".

→ *protoNameP*
Desired protocol to use, either "udp" or "tcp".

← *bufP*
Pointer to a `NetServInfoBufType` struct in which to store the results of the lookup.

→ *timeout*
Maximum timeout in system ticks; -1 means wait forever.

← *errP*
Contains an error code if the return value is 0.

Returns Returns a pointer to the `NetServInfoType` portion of `bufP` that contains results of the lookup. If the return value is 0, and error has occurred and `errP` contains one of the following values:

`0`
No error

`netErrTimeout`
Call timed out.

`netErrNotOpen`
The referenced net library has not been opened yet.

`netErrUnknownProtocol`

`netErrUnknownService`

Sockets Equivalent

```
struct servent *getservbyname (char *addr,
char *proto);
```

Comments This call is a convenience call for looking up a standard port number given the name of a service and the protocol to use.

See Also NetLibGetHostByName()

NetLibIFAttach Function

Purpose Attach a new network interface.

Declared In `NetMgr.h`

Prototype

```
Err NetLibIFAttach (UInt16 libRefNum,
   UInt32 ifCreator, UInt16 ifInstance,
   Int32 timeout)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *ifCreator*
Creator of interface to attach.

→ *ifInstance*
Instance number of interface to attach. The instance number is one of the values returned by NetLibIFGet().

→ *timeout*
Timeout in ticks; -1 means infinite timeout.

Returns Returns one of the following values:

`0`
Success.

`netErrInterfaceNotFound`

`netErrTooManyInterfaces`

Sockets Equivalent None

Comments This call can be used to attach a new network interface to the net library. Network interfaces are self-contained databases of type `'neti'`. The `ifCreator` parameter to this function is used to locate the network interface database of the given creator.

If the net library is already open when this call is made, the network interface's database will be located and then called to initialize itself and attach itself to the protocol stack in real time. If the net library is not open when this call is made, the creator and instance number of the interface are stored in the active configuration. You need to save the active configuration using `NetLibConfigSaveAs()` if you want the interface to be initialized and attached to the stack the next time the net library is opened.

See Also `NetLibIFGet()`, `NetLibIFDetach()`

NetLibIFDetach Function

Purpose Detach a network interface from the protocol stack.

Declared In `NetMgr.h`

Prototype

```
Err NetLibIFDetach (UInt16 libRefNum,
   UInt32 ifCreator, UInt16 ifInstance,
   Int32 timeout)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *ifCreator*
Creator of interface to detach.

→ *ifInstance*
Instance number of interface to detach.

→ *timeout*
Timeout in ticks; -1 means infinite timeout.

Returns Returns one of the following values:

`0`
Success.

`netErrInterfaceNotFound`

Sockets Equivalent None

Comments If the net library is already open when this call is made, the interface is brought down and detached from the protocol stack in real time. If the net library is not open when this call is made, the creator and instance number of the interface are removed from the active

configuration. You need to save the active configuration using NetLibConfigSaveAs() if you don't want the interface to be attached the next time the library is opened.

See Also NetLibIFGet(), NetLibIFAttach()

NetLibIFDown Function

Purpose Bring an interface down and hang up a connection.

Declared In NetMgr.h

Prototype

```
Err NetLibIFDown (UInt16 libRefNum,
   UInt32 ifCreator, UInt16 ifInstance,
   Int32 timeout)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *ifCreator*
Creator of interface to attach.

→ *ifInstance*
Instance number of interface to attach.

→ *timeout*
Timeout in ticks; -1 means wait forever.

Returns Returns one of the following values:

0
Success.

netErrNotOpen
The referenced net library has not been opened yet.

netErrInterfaceNotFound

Sockets Equivalent None

Comments The net library must be open before this call can be made. For dial-up interfaces, this call terminates a connection and hangs up the modem if necessary.

NetLibClose() automatically brings down any attached interfaces, so this routine doesn't normally have to be called.

If the interface is already down, this routine returns immediately with no error.

See Also NetLibIFGet(), NetLibIFAttach(), NetLibIFDetach(), NetLibIFUp()

NetLibIFGet Function

Purpose Get the creator and instance number of an installed interface by index.

Declared In `NetMgr.h`

Prototype

```
Err NetLibIFGet (UInt16 libRefNum, UInt16 index,
    UInt32 *ifCreatorP, UInt16 *ifInstanceP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *index*
Index of the interface to get. Indices start at 0.

← *ifCreatorP*
The interface's creator.

← *ifInstanceP*
The interface's instance number.

Returns Returns one of the following values:

`0`
Success.

`netErrInvalidInterface`
Index too high

`netErrPrefNotFound`
No current value for setting.

Sockets Equivalent None

Comments To get a list of all installed interfaces, call this function with successively increasing indices starting from 0 until the error `netErrInvalidInterface` is returned.

The `ifCreator` and `ifInstance` values returned from this call can then be used with the `NetLibSettingGet()` call to get more information about that particular interface.

See Also `NetLibIFAttach()`, `NetLibIFDetach()`, "`Settings for Interface Selection`" in the *Palm OS Programmer's Companion*, vol. II, *Communications*

NetLibIFSettingGet Function

Purpose Retrieves a network interface specific setting.

Declared In `NetMgr.h`

Prototype

```
Err NetLibIFSettingGet (UInt16 libRefNum,
   UInt32 ifCreator, UInt16 ifInstance,
   UInt16 setting, void *valueP,
   UInt16 *valueLenP)
```

Parameters

→ *`libRefNum`*
Reference number of the net library.

→ *`ifCreator`*
Creator of the network interface.

→ *`ifInstance`*
Instance number of the network interface.

→ *`setting`*
Setting to retrieve; one of the `NetIFSettingEnum` constants.

← *`valueP`*
Space for return value of setting.

↔ *`valueLenP`*
On entry, size of `valueP`. On exit, actual size of setting.

Returns Returns one of the following values:

`0`
Success.

`netErrUnknownSetting`
Invalid setting constant.

`netErrPrefNotFound`
No current value for setting.

`netErrBufTooSmall`
: `valueP` was too small to hold entire setting. Setting value was truncated to fit in `valueP`.

`netErrUnimplemented`

`netErrInterfaceNotFound`

`netErrBufWrongSize`

Sockets Equivalent

None

Comments

This call can be used to retrieve the current value of any network interface setting. The caller must pass a pointer to a buffer to hold the return value (valueP), the size of the buffer (`*valueLenP`), and the setting ID (`setting`). The setting ID is one of the constants in the `NetIFSettingEnum` type.

Some settings, such as the login script, are variable size. For these types of settings, you can obtain the actual size required for the buffer by passing 0 for `*valueLenP`. The required size is returned in `valueLenP`.

Table 66.2 lists the network interface settings and the size of each setting. Some are only applicable to certain types of interfaces. Settings not applicable to a specific interface can be safely ignored and not set to any particular value.

Table 66.2 Network Interface Settings

netIFSetting...	Type	Description
`ResetAll`	`void`	Use with `NetLibIFSettingSet()` only. This clears all other settings for the interface to their default values.
`Up`	`UInt8`	Read-only. `true` if interface is currently up.
`Name`	`Char[32]`	Read-only. Name of this interface.
`ReqIPAddr`	`UInt32`	IP address of interface.
`SubnetMask`	`UInt32`	Subnet mask for interface. Doesn't need to be specified for PPP or SLIP type connections.
`Broadcast`	`UInt32`	Broadcast address for interface. Doesn't need to be specified for PPP or SLIP type connections.
`Username`	`Char[32]`	User name. Only required if the login script uses the user name substitution escape sequence in it. Call `NetLibIFSettingSet()` with a `valueLen` of 0 to remove this setting.
`Password`	`Char[32]`	Password. Only required if the login script uses the password substitution escape sequence in it. Call `NetLibIFSettingSet()` with a `valueLen` of 0 to remove this setting. If the login script uses password substitution and no password setting is set, the user will be prompted for a password at connect time.
`AuthUsername`	`Char[32]`	Authentication user name. Only required if the authentication protocol uses a different user name than the what's in the `netIFSettingUsername` setting. If this setting is empty (`valueLen` of 0), the Username setting will be used instead. Call `NetLibIFSettingSet()` with a `valueLen` of 0 to remove this setting.

Table 66.2 Network Interface Settings *(continued)*

netIFSetting...	Type	Description
`AuthPassword`	`Char[32]`	Authentication password. If "$" then the user will be prompted for the authentication password at connect time. Else, if 0 length, then the `netIFSettingPassword` setting or the result of its prompt will be used instead. Call `NetLibIFSettingSet()` with a `valueLen` of 0 to remove this setting.
`ServiceName`	`Char[]`	Service name. Used for display purposes while showing the connection progress dialog box. Call `NetLibIFSettingSet()` with a `valueLen` of 0 to remove this setting.
`LoginScript`	`Char[]`	Login script. Only required if the particular service requires a login sequence. Call `NetLibIFSettingSet()` with a `valueLen` of 0 to remove this setting. See below for a description of the login script format.
`ConnectLog`	`Char[]`	Connect log. Generally, this setting is just retrieved, not set. It contains a log of events from the most recent login. To clear this setting, call `NetLibIFSettingSet()` with a `valueLen` of 0.
`InactivityTimeout`	`UInt16`	Maximum number of seconds of inactivity allowed. Set to 0 to ignore.
`EstablishmentTimeout`	`UInt16`	Maximum delay, in seconds, allowed between each stage of connection establishment or login script line. Must be non-zero.
`DynamicIP`	`UInt8`	If non-zero, negotiate for an IP address. If zero, the IP address specified in the `netIFSettingReqIPAddr` setting will be used. Default is `false`.
`VJCompEnable`	`UInt8`	If non-zero, enable VJ header compression. Default is `true` for PPP, `false` for SLIP, and `true` for CSLIP.

Table 66.2 Network Interface Settings *(continued)*

netIFSetting...	Type	Description
`VJCompSlots`	`UInt8`	Number of slots to use for VJ compression. Default is 4 for PPP and 16 for SLIP and CSLIP. More slots require more memory so it is best to keep this number to a minimum.
`MTU`	`UInt16`	Maximum transmission unit in octets. Currently not implemented in SLIP or PPP.
`AsyncCtlMap`	`UInt32`	Bit mask of characters to escape for PPP. Default is 0.
`PortNum`	`UInt16`	Which serial communication port to use. Port 0 is the only port available on the device.
`BaudRate`	`UInt32`	Serial port baud rate to use in bits per second.
`FlowControl`	`UInt8`	If bit 0 is 1, use hardware handshaking on the serial port. Default is no hardware handshaking.
`StopBits`	`UInt8`	Number of stop bits. Default is 1.
`ParityOn`	`UInt8`	`true` if parity detection enabled. Default is `false`.
`ParityEven`	`UInt8`	`true` for even parity detection. Default is `true`.
`UseModem`	`UInt8`	If `true`, dial-up through modem. If `false`, go direct over serial port
`PulseDial`	`UInt8`	If `true`, pulse dial modem. Else, tone dial. Default is tone dial.
`ModemInit`	`Char[]`	Zero-terminated modem initialization string, not including the "AT". If not specified (`valueLen` of 0), the modem initialization string from system preferences are used.
`ModemPhone`	`Char[]`	Zero-terminated modem phone number string. Only required if `netIFSettingUseModem` is true.

Table 66.2 Network Interface Settings *(continued)*

netIFSetting...	Type	Description
`RedialCount`	`UInt16`	Number of times to re-dial modem when trying to establish a connection. Only required if `netIFSettingUseModem` is true.
`DNSQuery`	`UInt8`	`true` if PPP queries for DNS address. The default is `true`.
`TraceBits`	`UInt32`	A bitfield of various trace bits. See "`Tracing Bits`." An application can get a list of events in the trace buffer using the `NetLibMaster()` call. Each interface has its own trace bits setting so that trace event recording in each interface can be selectively enabled or disabled.
`ActualIPAddr`	`UInt32`	Read-only. The actual IP address that the interface ends up using. The login script execution engine stores the result of the "g" (get IP address) command here as does the PPP negotiation logic.
`ServerIPAddr`	`UInt32`	Read-only. The IP address of the PPP server we're connected to.
`BringDownOnPowerDown`	`UInt8`	`true` if the interface is brought down when the Palm OS device is turned off.
`RawMode`	`UInt32`	Specifies if the interface is in raw mode. The net library places an interface in raw mode when it is bound to a raw socket in the raw domain. Raw sockets are available in Palm OS version 3.0 and higher.
DriverVersion	Char[20]	Read-only. The version number of the network interface device driver. This setting is defined only if 5.1 New Feature Set is present.
FirmwareVersion	Char[20]	Read-only. The firmware version of the network interface device, if any. This setting is defined only if 5.1 New Feature Set is present.

Table 66.2 Network Interface Settings *(continued)*

netIFSetting...	Type	Description
FirmwareDate	UInt32	Read-only. Firmware date in seconds since midnight, January 1, 1904. This setting is defined only if 5.1 New Feature Set is present.
80211Device	UInt8	Read-only. Indicates whether or not the interface supports IEEE 802.11 wireless networking. This setting is defined only if 5.1 New Feature Set is present.
80211ESSID	Char[32]	For IEEE 802.11 interfaces only. The ESS ID of the radio. This setting is defined only if 5.1 New Feature Set is present.
80211AccessPointBSSID	UInt8[6]	Read-only. For IEEE 802.11 interfaces only. The BSS ID (MAC address) of the access point to which the radio is connected. This setting is defined only if 5.1 New Feature Set is present.
80211AssociationStatus	UInt8	Read-only. For IEEE 802.11 interfaces only. `true` if the radio is associated. This setting is defined only if 5.1 New Feature Set is present.
80211MKKCallSign	Char[15]	Read-only. For IEEE 802.11 interfaces with radios programmed for operation in Japan only. The MKK call sign. This setting is defined only if 5.1 New Feature Set is present.
80211CountryText	Char[34]	Read-only. For IEEE 802.11 interfaces only. The radio's country code, which the radio uses to check if it operates within a particular country's regulations. This setting is defined only if 5.1 New Feature Set is present.

See Also `NetLibIFSettingSet()`, `NetLibSettingGet()`, `NetLibSettingSet()`, "Interface Specific Settings" in the *Palm OS Programmer's Companion*, vol. II, *Communications*

NetLibIFSettingSet Function

Purpose Sets a network interface specific setting.

Declared In `NetMgr.h`

Prototype

```
Err NetLibIFSettingSet (UInt16 libRefNum,
   UInt32 ifCreator, UInt16 ifInstance,
   UInt16 setting, void *valueP, UInt16 valueLen)
```

Parameters

→ *`libRefNum`*
Reference number of the net library.

→ *`ifCreator`*
Creator of the network interface.

→ *`ifInstance`*
Instance number of the network interface.

→ *`setting`*
The setting to set, one of the `NetIFSettingEnum` constants. See Table 66.2.

→ *`valueP`*
Space new value of setting.

→ *`valueLen`*
Size of new setting.

Returns Returns one of the following values:

`0`
Success.

`netErrUnknownSetting`
Invalid setting constant.

`netErrPrefNotFound`
No current value for setting.

`netErrUnimplemented`

`netErrInterfaceNotFound`

`netErrBufWrongSize`

`netErrReadOnlySetting`

Sockets Equivalent None

Comments This call can be used to set the current value of any network interface setting. The caller must pass a pointer to a buffer which holds the new value (`valueP`), the size of the buffer (`valueLen`), and the setting ID (`setting`).

See `NetLibIFSettingGet()` for an explanation of each of the settings.

Of particular interest is the `netIFSettingResetAll` setting, which, if used, resets all settings for the interface to their default values. When using this setting, `valueP` and `valueLen` are ignored.

See Also `NetLibIFSettingGet()`, `NetLibSettingGet()`, `NetLibSettingSet()`, "Interface Specific Settings" in the *Palm OS Programmer's Companion*, vol. II, *Communications*

NetLibIFUp Function

Purpose Bring an interface up and establish a connection.

Declared In `NetMgr.h`

Prototype

```
Err NetLibIFUp (UInt16 libRefNum,
   UInt32 ifCreator, UInt16 ifInstance)
```

Parameters

→ *`libRefNum`*
Reference number of the net library.

→ *`ifCreator`*
Creator of interface to attach.

→ *`ifInstance`*
Instance number of interface to attach.

Returns Returns one of the following values:

`0`
Success.

`netErrNotOpen`
The referenced net library has not been opened yet.

`netErrInterfaceNotFound`

`netErrUserCancel`

`netErrBadScript`

`netErrPPPTimeout`

`netErrAuthFailure`

`netErrPPPAddressRefused`

Sockets Equivalent None

Comments The net library must be open before this call can be made. For dial-up interfaces, this call will dial up the modem if necessary and run through the connect script to establish the connection.

If the interface is already up, this routine returns immediately with no error. This call doesn't take a timeout parameter because it relies on each interface to have its own established timeout setting.

See Also `NetLibIFGet()`, `NetLibIFAttach()`, `NetLibIFDetach()`, `NetLibIFDown()`

NetLibMaster Function

Purpose Retrieves the network statistics, interface statistics, and the contents of the trace buffer.

Declared In `NetMgr.h`

Prototype

```
Err NetLibMaster (UInt16 libRefNum, UInt16 cmd,
   NetMasterPBPtr pbP, Int32 timeout)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *cmd*
Function to perform (`NetMasterEnum` type). The following commands are supported:

`netMasterInterfaceInfo`

`netMasterInterfaceStats`

`netMasterIPStats`

`netMasterICMPStats`

`netMasterUDPStats`

`netMasterTCPStats`

`netMasterTraceEventGet`

↔ *pbP*
: Command parameter block.

→ *timeout*
: Timeout in ticks; -1 means wait forever.

Returns Returns one of the following values:

`0`
: No error

`netErrNotOpen`
: The referenced net library has not been opened yet.

`netErrParamErr`

`netErrUnimplemented`

Sockets Equivalent None

Comments This call allows applications to get detailed information about the net library. This information is usually helpful in debugging network configuration problems.

This function takes a command word (`cmd`) and parameter block pointer (`pbP`) as arguments and returns its results in the parameter block on exit. Which values you must specify in the parameter block and which values are returned are specific to the command you specify.

netMasterInterfaceInfo

The `pbP->interfaceInfo` struct specifies interface information.

→	`index`	Index of interface to fetch info about.
←	`creator`	Creator of interface.
←	`instance`	Instance of interface.
←	`netIFP`	Private interface info pointer.
←	`drvrName`	Driver type that interface uses (“PPP”, “SLIP”, etc.).
←	`hwName`	Hardware driver name (“Serial Library”, etc.).

←	`localNetHdrLen`	Number of bytes in local net header.
←	`localNetTrailerLen`	Number of bytes in local net trailer.
←	`localNetMaxFrame`	Local net maximum frame size.
←	`ifName`	Interface name with instance number concatenated.
←	`driverUp`	`true` if interface driver is up.
←	`ifUp`	`true` if interface media layer is up.
←	`hwAddrLen`	Length of interface's hardware address.
←	`hwAddr`	Interface's hardware address.
←	`mtu`	Maximum transfer unit of interface.
←	`speed`	Speed in bits per second.
←	`lastStateChange`	Time in milliseconds of last state change.
←	`ipAddr`	IP address of interface.
←	`subnetMask`	Subnet mask of local network.
←	`broadcast`	Broadcast address of local network.

netMasterInterfaceStats

The `pbP->interfaceStats` structure specifies interface statistics.

→	`index`	Index of interface to fetch info about.
←	`inOctets`	Number of octets received.
←	`inUcastPkts`	Number of packets received.
←	`inNUcastPkts`	Number of broadcast packets received.

←	`inDiscards`	Number of incoming packets that were discarded.
←	`inErrors`	Number of packet errors encountered.
←	`inUnknownProtos`	Number of unknown protocols encountered.
←	`outOctets`	Number octets sent.
←	`outUcastPkts`	Number of packets sent.
←	`outNUcastPkts`	Number of broadcast packets sent.
←	`outDiscards`	Number of packets discarded.
←	`outErrors`	Number of outbound packet errors.

netMasterIPStats

The `pbP->ipStats` structure contains statistics about the IP protocol. See `NetMgr.h` for a complete list of statistics returned.

netMasterICMPStats

The `pbP->icmpStats` structure contains statistics about the ICMP protocol. See `NetMgr.h` for a complete list of statistics returned.

netMasterUDPStats

The `pbP->udpStats` structure contains statistics about the UDP protocol. See `NetMgr.h` for a complete list of statistics returned.

netMasterTCPStats

The `pbP->tcpStats` structure contains statistics about the TCP protocol. See `NetMgr.h` for a complete list of statistics returned.

netMasterTraceEventGet

The `pbP->traceEventGet` structure contains a trace event.

→ `index` Index of event to fetch.

← `textP` Pointer to text string to return event in. Should be at least 256 bytes long.

See Also `NetLibSettingSet()`

NetLibOpen Function

Purpose Opens and initializes the net library.

Declared In `NetMgr.h`

Prototype

```
Err NetLibOpen (UInt16 libRefnum,
   UInt16 *netIFErrsP)
```

Parameters

→ *`libRefnum`*
Reference number of the net library.

← *`netIFErrsP`*
First error encountered when bringing up network interfaces. (See `NetLibIFUp()` for a list of possible values.)

Returns Returns one of the following values:

`0`
No error.

`netErrAlreadyOpen`
Not really an error; returned if library was already open and the open count was simply incremented.

`netErrOutOfMemory`
Not enough memory available to open the library.

`netErrNoInterfaces`
Incorrect setup.

`netErrPrefNotFound`
Incorrect setup.

Comments Applications must call this function before using the net library. If the net library was already open, `NetLibOpen` increments its open count. Otherwise, it opens the library, initializes it, starts up the net protocol stack component of the library as a separate task, and brings up all attached network interfaces.

`NetLibOpen` uses settings saved in the net library's preferences database during initialization. These settings include the interfaces to attach, the IP addresses, etc. It's assumed that these settings have been previously set up by a preference panel or equivalent so an application doesn't normally have to set them up before calling `NetLibOpen`.

If any of the attached interfaces fails to come up, `*netIFErrsP` will contain the error number of the first interface that encountered a problem.

Compatibility `NetLibOpen` behaves slightly differently in version 3.2 and later than it does in previous releases. In version 3.2 and later, `NetLibOpen` calls `NetLibOpenConfig()` specifying the default configuration. `NetLibOpenConfig` reverts all settings to their saved, default values before opening the net library.

See Also `SysLibFind()`, `NetLibClose()`, `NetLibOpenCount()`

NetLibOpenConfig Function

Purpose Opens and initializes the net library with the specified configuration.

Prototype

```
Err NetLibOpenConfig (UInt16 refNum,
   UInt16 configIndex, UInt32 openFlags,
   UInt16 *netIFErrP)
```

Parameters

→ *refNum*
Reference number of the net library.

→ *configIndex*
Index of the configuration to use. 0 means use the default configuration as defined in the Network preferences panel.

→ *openFlags*
Not used. Pass 0 for this parameter.

← *netIFErrP*
Pointer to return error code for interfaces.

Returns Returns one of the following values:

0
No error.

`memErrNotEnoughSpace`

`netErrAlreadyOpen`
Not really an error; returned if library was already open and the open count was simply incremented.

`netErrAlreadyOpenWithAnotherConfig`
Another application has the net library open with a configuration that is incompatible with the one specified.

`netErrBufTooSmall`

`netErrConfigAliasErr`
A configuration alias was specified, but the alias could not be resolved.

`netErrConfigCantDelete`

`netErrConfigEmpty`
The configuration is not defined.

`netErrConfigNotFound`
The specified configuration index is invalid.

`netErrInterfaceNotFound`

`netErrOutOfCmdBlocks`

`netErrOutOfMemory`
Not enough memory available to open the library.

`netErrNoInterfaces`
Incorrect setup.

`netErrParamErr`

`netErrPrefNotFound`
Incorrect setup.

`netErrTimeout`

Sockets Equivalent

None

Comments

Use this routine instead of `NetLibOpen()` when you want to open the net library with a non-default configuration. If the default net library configuration is not suitable for your application, you may use one of the built-in aliases to specify a configuration that is suitable (see "Configuration Aliases").

`NetLibOpenConfig` tries to open the net library and initialize it with the specified configuration. If another application has the net library open with an incompatible configuration, it returns an error. If the net library is in the close-wait state, this function completely closes the net library and then reopens it using the new configuration. If the net library can be opened with the new configuration, `NetLibOpenConfig` first saves the current configuration so that it can be restored when your application closes the net library.

Typically, applications use the `NetLibConfigList()` function to obtain the list of available configurations and present this list to the user. Then they call `NetLibConfigIndexFromName()` with the user's selection to get the index of the configuration that the user selected.

The constant `netConfigIndexCurSettings` specifies the current configuration. You can specify `netConfigIndexCurSettings` as the `configIndex` for testing purposes.

Compatibility Implemented only if 3.2 New Feature Set is present.

See Also `NetLibOpen()`, `SysLibFind()`, `NetLibClose()`, `NetLibOpenCount()`

NetLibOpenCount Function

Purpose Retrieves the open count of the net library.

Declared In `NetMgr.h`

Prototype

```
Err NetLibOpenCount (UInt16 refNum,
   UInt16 *countP)
```

Parameters

→ *refNum*
Reference number of the net library.

← *countP*
Contains the open count of the net library upon return.

Returns `Always returns 0.`

Sockets Equivalent None.

Comments This call will most likely only be used by the Network preferences panel. Most applications will simply call `NetLibOpen()` unconditionally during startup and `NetLibClose()` when they exit.

NetLibReceive Function

Purpose Receive data from a socket into a single buffer.

Declared In `NetMgr.h`

Prototype

```
Int16 NetLibReceive (UInt16 libRefNum,
   NetSocketRef socket, void *bufP,
   UInt16 bufLen, UInt16 flags, void *fromAddrP,
   UInt16 *fromLenP, Int32 timeout, Err *errP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *socket*
Descriptor for the open socket.

← *bufP*
Pointer to buffer to hold received data.

→ *bufLen*
Length of `bufP` buffer.

→ *flags*
One or more `netIOFlagxxx` flags. See "`I/O Flags`."

← *fromAddrP*
Pointer to buffer to hold address of sender (a `NetSocketAddrType`).

↔ *fromLenP*
On entry, size of `fromAddrP` buffer. On exit, actual size of returned address in `fromAddrP`.

→ *timeout*
Maximum timeout in system ticks; -1 means wait forever.

← *errP*
Contains an error code if the return value is -1.

Returns Returns the number of bytes successfully received. If the return value is 0, the socket has been shut down by the remote host. If the

return value is -1, an error has occurred, and `errP` contains one of the following values:

`0`
: No error.

`netErrTimeout`
: Call timed out.

`netErrNotOpen`
: The referenced net library has not been opened yet.

`netErrParamErr`

`netErrSocketNotOpen`

`netErrWouldBlock`

`netErrUserCancel`

Sockets Equivalent

```
int recvfrom (int socket, const void *bufP,
int bufLen, int flags, const void *fromAddrP,
int *fromLenP);
```

```
int recv (int socket, const void *bufP,
int bufLen, int flags);
```

```
int read (int socket, const void *bufP,
int bufLen);
```

Comments

For stream-based sockets, this call reads whatever bytes are available and returns the number of bytes actually read into the caller's buffer. If there is no data available, this call will block until at least one byte arrives, until the socket is shut down by the remote host, or until a timeout occurs.

For datagram-based sockets, this call reads a complete datagram and returns the number of bytes in the datagram. If the caller's buffer is not large enough to hold the entire datagram, the end of the datagram is discarded. If a datagram is not available, this call will block until one arrives, or until the call times out.

The data is read into a single buffer pointed to by `bufP`.

See Also

`NetLibReceive()`, `NetLibDmReceive()`, `NetUReadN`, `NetLibSend()`, `NetLibSendPB()`

NetLibReceivePB Function

Purpose Receive data from a socket into a multi-buffer gather-read array.

Declared In `NetMgr.h`

Prototype

```
Int16 NetLibReceivePB (UInt16 libRefNum,
   NetSocketRef socket, NetIOParamType *pbP,
   UInt16 flags, Int32 timeout, Err *errP)
```

Parameters

→ *libRefNum*
: Reference number of the net library.

→ *socket*
: Descriptor for the open socket.

→ *pbP*
: Pointer to parameter block containing buffer info.

→ *flags*
: One or more `netIOFlagxxx` flags. See "I/O Flags."

→ *timeout*
: Maximum timeout in system ticks; -1 means wait forever.

← *errP*
: Contains an error code if the return value is -1.

Returns Returns the number of bytes successfully received. Returns 0 if the socket has been shut down by the remote host. If the return value is -1, an error has occurred, and `errP` contains one of the following values:

`0`
: No error.

`netErrTimeout`
: Call timed out.

`netErrNotOpen`
: The referenced net library has not been opened yet.

`netErrParamErr`

`netErrSocketNotOpen`

`netErrWouldBlock`

Sockets Equivalent

```
int recvmsg (int socket, const struct msghdr *pbP,
int flags);
```

Comments The `pbP` parameter is a pointer to a `NetIOParamType` structure. `NetIOParamType` is defined as follows:

Prototype

```
typedef struct {
  UInt8 *addrP;
  UInt16 addrLen;
  NetIOVecPtr iov;
  UInt16 iovLen;
  UInt8 *accessRights;
  UInt16 accessRightsLen;
} NetIOParamType, *NetIOParamPtr;
```

Fields

`addrP`
: Address of sender, set by `NetLibReceivePB`. Set to 0 if you don't require this field.

`addrLen`
: Length of `*addrP`.

`iov`
: Array of buffers into which the data should be received. `NetIOVecPtr` is a pointer to a `NetIOVecType` structure, which has two fields:

  `bufP`
  : Pointer to a buffer.

  `bufLen`
  : Length of `bufP`.

`iovLen`
: Length of the `iov` array.

`accessRights`
: Access rights. This field currently isn't used and should be set to 0.

`accessRightsLen`
: Length of the `*accessRights`. This field currently isn't used and should be set to 0.

For stream-based sockets, this call reads whatever bytes are available and returns the number of bytes actually read into the caller's buffer. If no data is available, this call will block until at least one byte arrives, until the socket is shut down by the remote host, or until a timeout occurs.

For datagram-based sockets, this call reads a complete datagram and returns the number of bytes in the datagram. If the caller's buffer is not large enough to hold the entire datagram, the end of the datagram is discarded. If a datagram is not available, this call will block until one arrives, or until the call times out.

The data is read into the gather-read array specified by the `pbP->iov` array.

See Also `NetLibReceive()`, `NetLibDmReceive()`, `NetLibSend()`, `NetLibSendPB()`

NetLibSelect Function

Purpose Blocks until I/O is ready on one or more descriptors, where a descriptor can represent socket input, socket output, or a user input event like a pen tap or key press.

Declared In `NetMgr.h`

Prototype

```
Int16 NetLibSelect (UInt16 libRefNum,
   UInt16 width, NetFDSetType *readFDs,
   NetFDSetType *writeFDs,
   NetFDSetType *exceptFDs,
   Int32 timeout, Err *errP)
```

Parameters

→ *libRefNum*
: Reference number of the net library.

→ *width*
: Number of descriptor bits to check in the `readFDs`, `writeFDs`, and `exceptFDs` descriptor sets.

↔ *readFDs*
: Pointer to 32-bit `NetFDSetType` containing set of bits representing descriptors to check for input.

↔ *writeFDs*
: Pointer to 32-bit `NetFDSetType` containing set of bits representing descriptors to check for output.

↔ *exceptFDs*
: Pointer to 32-bit `NetFDSetType` containing set of bits representing descriptors to check for exception conditions. This parameter is ignored. Upon return, its bits are always cleared.

→ *timeout*
Maximum timeout in system ticks; -1 means wait forever.

← *errP*
Contains an error code if the return value is -1.

Returns Returns the sum total number of ready file descriptors in `*readFDs`, `*writeFDs`, and `*exceptFDs`. Returns 0 upon timeout. If the return value is -1, an error has occurred, and `errP` contains one of the following values:

`0`
No error

`netErrTimeout`
Call timed out.

`netErrNotOpen`
The referenced net library has not been opened yet.

Sockets Equivalent

```
int select (int width, fd_set *readfds,
fd_set *writefds, fd_set *exceptfds,
struct timeval *timeout);
```

Comments This call blocks until one or more descriptors are ready for I/O. In the Palm OS environment, a descriptor is either a `NetSocketRef` or the "stdin" descriptor, `sysFileDescStdIn`. The `sysFileDescStdIn` descriptor will be ready for input whenever a user event is available like a pen tap or key press.

The caller should set which bits in each descriptor set need to be checked by using the `netFDZero` and `netFDSet` macros. After this call returns, the macro `netFDIsSet` can be used to determine which descriptors in each set are actually ready.

On exit, the total number of ready descriptors is returned and each descriptor set is updated with the appropriate bits set for each ready descriptor in that set.

The following example illustrates how to use this call to check for input on a socket or a user event:

```
Err     err;
NetSocketRef    socket;
NetFDSetType    readFDs,writeFDs,exceptFDs;
Int16       numFDs;
UInt16      width;
```

```
// Create the descriptor sets
netFDZero(&readFDs);
netFDZero(&writeFDs);
netFDZero(&exceptFDs);
netFDSet(sysFileDescStdIn, &readFDs);
netFDSet(socket, &readFDs);

// Calculate the max descriptor number and
// use that +1 as the max width.
// Alternatively, we could simply use the
// constant netFDSetSize as the width which
// is simpler but makes the NetLibSelect call
// slightly slower.
width = sysFileDescStdIn;
if (socket > width) width = socket;

// Wait for any one of the descriptors to be
// ready.
numFDs = NetLibSelect(AppNetRefnum, width+1,
  &readFDs, &writeFDs, &exceptFDs,
  AppNetTimeout, &err);
```

Also see the `NetSample` example application in the `Palm OS Examples` folder. The function `CmdTelnet` in the file `CmdTelnet.c` shows how to use the Berkeley sockets `select` function and how to interpret the results.

See Also `NetLibSocketOptionSet()`

NetLibSend Function

Purpose Send data to a socket from a single buffer.

Declared In `NetMgr.h`

Prototype

```
Int16 NetLibSend (UInt16 libRefNum,
   NetSocketRef socket, void *bufP,
   UInt16 bufLen,
   UInt16 flags, void *toAddrP, UInt16 toLen,
   Int32 timeout, Err *errP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *socket*
Descriptor for the open socket.

→ *bufP*
Pointer to data to write.

→ *bufLen*
Length of data to write

→ *flags*
One or more `netIOFlagxxx` flags. See "I/O Flags."

→ *toAddrP*
Address to send to (a pointer to a `NetSocketAddrType`), or 0.

→ *toLen*
Size of `toAddrP` buffer.

→ *timeout*
Maximum timeout in system ticks; -1 means wait forever.

← *errP*
Contains an error code if the return value is -1.

Returns Returns the number of bytes successfully sent. Returns 0 if the socket has been shut down by the remote host. If the return value is -1, an error has occurred, and `errP` contains one of the following values:

0
No error.

`netErrTimeout`
Call timed out.

`netErrNotOpen`
The referenced net library has not been opened yet.

`netErrParamErr`

`netErrSocketNotOpen`

`netErrMessageTooBig`

`netErrSocketNotConnected`

`netErrSocketClosedByRemote`

`netErrIPCantFragment`

```
netErrIPNoRoute
netErrIPNoSrc
netErrIPNoDst
netErrIPktOverflow
netErrOutOfCmdBlocks
netErrOutOfPackets
netErrInterfaceNotFound
netErrInterfaceDown
netErrUnreachableDest
netErrNoMultiPktAddr
netErrWouldBlock
```

Sockets Equivalent

```
int sendto (int socket, const void *bufP,
int bufLen, int flags, const void *toAddrP,
int toLen);
```

```
int send (int socket, const void *bufP,
int bufLen, int flags);
```

```
int write (int socket, const void *bufP,
int bufLen,);
```

Comments

This call attempts to write data to the specified socket and returns the number of bytes actually sent, which may be less than or equal to the requested number of bytes. The data is passed in a single buffer that `bufP` points to.

For datagram sockets, you must only send a single packet at a time. If the data is too large to fit in a single UDP packet (1536 bytes), no data is sent and -1 is returned.

The `toAddrP` field applies only to datagram sockets without an existing connection. An error is returned if the datagram socket was previously connected and `toAddrP` is specified. Stream-based sockets, by definition, must have a connection established with a remote host before data can be written. Raw sockets (supported in Palm OS version 3.0 and higher) must construct the entire IP header, including the destination address, before data can be sent; thus, the address is taken from the data to be sent.

If there isn't enough buffer space to send any data, this call will block until there is enough buffer space, or until a timeout.

NOTE: For stream-based sockets, this call may write only a portion of the desired data. It always returns the number of bytes actually written. Consequently, the caller should be prepared to call this routine repeatedly until the desired number of bytes have been written, or until it returns 0 or -1.

See Also NetLibSendPB(), NetUWriteN(), NetLibReceive(), NetLibReceivePB(), NetLibDmReceive()

NetLibSendPB Function

Purpose Send data to a socket from a scatter-write array.

Declared In `NetMgr.h`

Prototype

```
Int16 NetLibSendPB (UInt16 libRefNum,
   NetSocketRef socket, NetIOParamType *pbP,
   UInt16 flags, Int32 timeout, Err *errP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *socket*
Descriptor for the open socket.

→ *pbP*
Pointer to parameter block containing buffer info. See the description in NetLibReceivePB().

→ *flags*
One or more `netIOFlagxxx` flags. See "I/O Flags."

→ *timeout*
Maximum timeout in system ticks; -1 means wait forever.

← *errP*
Contains an error code if the return value is -1.

Returns Returns the number of bytes successfully sent. Returns 0 if the socket has been shut down by the remote host. If the return value is -1, an error has occurred, and `errP` contains one of the following values:

`0`
: No error.

`netErrTimeout`
: Call timed out.

`netErrNotOpen`
: The referenced net library has not been opened yet.

`netErrParamErr`

`netErrSocketNotOpen`

`netErrMessageTooBig`

`netErrSocketNotConnected`

`netErrSocketClosedByRemote`

`netErrIPCantFragment`

`netErrIPNoRoute`

`netErrIPNoSrc`

`netErrIPNoDst`

`netErrIPktOverflow`

`netErrOutOfCmdBlocks`

`netErrOutOfPackets`

`netErrInterfaceNotFound`

`netErrInterfaceDown`

`netErrUnreachableDest`

`netErrNoMultiPktAddr`

`netErrWouldBlock`

Sockets Equivalent

```
int sendmsg (int socket, const struct msghdr *pbP,
int flags);
```

Comments

This call attempts to write data to the given socket and returns the number of bytes actually sent, which may be less than or equal to the requested number of bytes. The data is passed in the scatter-write array specified in the `pbP` parameter block.

For datagram sockets, you must only send a single packet at a time. If the data is too large to fit in a single UDP packet, no data is sent and -1 is returned.

The `toAddrP` field applies only to datagram sockets without an existing connection. An error is returned if the datagram socket was previously connected and `toAddrP` is specified. Stream-based sockets, by definition, must have a connection established with a remote host before data can be written. Raw sockets (supported in Palm OS version 3.0 and higher) must construct the entire IP header, including the destination address, before data can be sent; thus, the address is taken from the data to be sent.

If there isn't enough buffer space to send any data, this call will block until there is space, or until a timeout.

NOTE: For stream-based sockets, this call may write only a portion of the desired data. It always returns the number of bytes actually written. Consequently, the caller should be prepared to call this routine repeatedly until the desired number of bytes have been written, or until it returns 0 or -1.

See Also `NetLibSend()`, `NetLibReceive()`, `NetLibReceivePB()`, `NetLibDmReceive()`

NetLibSettingGet Function

Purpose Retrieves a general setting.

Declared In `NetMgr.h`

Prototype

```
Err NetLibSettingGet (UInt16 libRefNum,
   UInt16 setting, void *valueP,
   UInt16 *valueLenP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *setting*
Setting to retrieve, one of the `NetSettingEnum` constants.

← *valueP*
Space for return value of setting.

↔ *valueLenP*
On entry, size of `valueP`. On exit, actual size of setting.

Returns Returns one of the following values:

`0`
Success.

`netErrUnknownSetting`
Invalid setting constant

`netErrPrefNotFound`
No current value for setting

`netErrBufTooSmall`
`valueP` was too small to hold entire setting. Setting value was truncated to fit in `valueP`.

`netErrBufWrongSize`

Sockets Equivalent None

Comments This call retrieves the current value of any general setting. The caller must pass a pointer to a buffer to hold the return value (`valueP`), the size of the buffer (*`valueLenP`), and the setting ID (`setting`). The setting ID is one of the `NetSettingEnum` constants in the `netSettingEnum` type.

Some settings, such as the host table, are variable size. For these types of settings, you can obtain the actual size required for the buffer by passing 0 for `*valueLenP`. The required size is returned in `valueLenP`.

Table 66.3 lists the general settings and the type of each setting.

Table 66.3 Net Library General Settings

netSetting...	Type	Description
`ResetAll`	`void`	Used for `NetLibSettingSet()` only. This will clear all other settings to their default values.
`PrimaryDNS`	`UInt32`	IP address of primary DNS server. This setting must be set to a non-zero IP address in order to support any of the name lookup calls.
`SecondaryDNS`	`UInt32`	IP address of secondary DNS server. Set to 0 to have stack ignore this setting.
`DefaultRouter`	`UInt32`	IP address of default router. Default value is 0 which is appropriate for most implementations with only one attached interface (besides loopback). Packets with destination IP addresses that don't lie in the subnet of an attached interface will be sent to this router through the default interface specified by the `netSettingDefaultIFCreator/ netSettingDefaultIFInstance` pair.
`DefaultIFCreator`	`UInt32`	Creator of the default network interface. Default value is 0, which is appropriate for most implementations. Packets with destination IP addresses that don't lie in the subnet of a directly attached interface are sent through this interface. If this setting is 0, the stack automatically makes the first non-loopback interface the default interface.
`DefaultIFInstance`	`UInt16`	Instance number of the default network interface. Packets with destination IP addresses that don't lie in the subnet of an attached interface are sent through the default interface. Default value is 0.
`HostName`	`Char[]`	A zero-terminated character string of 64 bytes or less containing the host name of this machine. This setting is not actually used by the stack. It's present mainly for informative purposes and to support the `gethostname/sethostname` sockets API calls. To clear the host name, call `NetLibIFSettingSet()` with a valueLen of 0.

Table 66.3 Net Library General Settings *(continued)*

netSetting...	Type	Description
`DomainName`	`Char[]`	A zero-terminated character string of 256 bytes or less containing the default domain. This default domain name is appended to all host names before name lookups are performed. If the name is not found, the host name is looked up again without appending the domain name to it. To have the stack not use the domain name, call `NetLibIFSettingSet()` with a `valueLen` of 0.
`HostTbl`	`Char[]`	A null-terminated character string containing the host table. This table is consulted first before sending a DNS query to the DNS server(s). To have the stack not use a host table, call `NetLibIFSettingSet()` with a `valueLen` of 0. The format of a host table is a series of lines separated by '\n' in the following format: `host.company.com A 111.222.333.444`
`CloseWaitTime`	`UInt32`	The close-wait time in milliseconds. This setting must be specified. See the discussion of the `NetLibClose()` call for an explanation of the close-wait time.
`TraceBits`	`UInt32`	A bitfield of various trace bits. See "`Tracing Bits`." Default value is (`netTracingErrors | netTracingAppMsgs`). An application can get a list of events in the trace buffer using the `NetLibMaster()` call.
`TraceSize`	`UInt32`	Maximum trace buffer size in bytes. Setting this setting always clears the existing trace buffer. Default is 2 KB.
`TraceRoll`	`UInt8`	Boolean value, default is `true` (non-zero). If `true`, trace buffer will roll over when it fills. If `false`, tracing will stop as soon as trace buffer fills.

See Also `NetLibSettingSet()`, `NetLibIFSettingSet()`, `NetLibIFSettingGet()`, `NetLibMaster()`

NetLibSettingSet Function

Purpose Sets a general setting.

Declared In `NetMgr.h`

Prototype

```
Err NetLibSettingSet (UInt16 libRefNum,
   UInt16 setting, void *valueP, UInt16 valueLen)
```

Parameters

→ *`libRefNum`*
Reference number of the net library.

→ *`setting`*
Setting to set; one of the `NetSettingEnum` constants. See Table 66.3.

→ *`valueP`*
New value for the setting.

→ *`valueLen`*
Size of new setting.

Returns Returns one of the following values:

`0`
Success.

`netErrUnknownSetting`
Invalid setting constant.

`netErrInvalidSettingSize`
`valueLen` was invalid for the given setting.

`netErrBufWrongSize`

`netErrReadOnlySetting`

Sockets Equivalent None

Comments This call can be used to set the current value of any general setting. The caller must pass a pointer to a buffer which holds the new value (`valueP`), the size of the buffer (`valueLen`), and the setting ID (`setting`). The setting ID is one of the `netSettingXXX` constants in the `NetSettingEnum` type.

If the net library is not open at the time this call is made, the setting is stored in the active configuration. You need to save the active configuration using `NetLibConfigSaveAs()` if you want the new

value of the setting to be used the next time the net library is opened.

See `NetLibSettingGet()` for an explanation of each of the settings.

Of particular interest is the `netSettingResetAll` setting, which, if used, will reset all general settings to their default values. When using this setting, `valueP` and `valueLen` are ignored.

See Also `NetLibSettingGet()`, `NetLibSettingSet()`, `NetLibIFSettingSet()`, `NetLibMaster()`

NetLibSocketAccept Function

Purpose Accept a connection from a stream-based socket.

Declared In `NetMgr.h`

Prototype

```
Int16 NetLibSocketAccept (UInt16 libRefnum,
   NetSocketRef socket,
   NetSocketAddrType *sockAddrP,
   Int16 *addrLenP, Int32 timeout, Err *errP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *socket*
Descriptor for the open socket.

← *sockAddrP*
Address of remote host is returned here.

↔ *addrLenP*
On entry, length of `sockAddrP` buffer in bytes. On exit, length of returned address stored in `*sockAddrP`.

→ *timeout*
Maximum timeout in system ticks; -1 means wait forever.

← *errP*
Contains an error code if the return value is -1.

Returns Returns the `NetSocketRef` of the new socket. If the return value is -1, an error has occurred, and `errP` contains one of the following values:

`0`
: No error.

`netErrTimeout`
: Call timed out.

`netErrNotOpen`
: The referenced net library has not been opened yet.

`netErrParamErr`

`netErrSocketNotOpen`

`netErrSocketNotConnected`

`netErrSocketClosedByRemote`

`netErrWrongSocketType`

`netErrSocketNotListening`

`netErrUnimplemented`

Sockets Equivalent

```
int accept (int socket, void *sockAddrP,
int *addrLenP);
```

Comments

Accepts the next connection request from a remote client. This call is only applicable to stream-based sockets. Before calling `NetLibSocketAccept` on a socket, a server application needs to:

- Open the socket (`NetLibSocketOpen()`).
- Bind the socket to a local address (`NetLibSocketBind()`).
- Set the maximum pending connection-request queue length (`NetLibSocketListen()`).

`NetLibSocketAccept` will block until a successful connection request is obtained from a remote client. After a successful connection is made, this call returns with the address of the remote host in `*sockAddrP` and the socket descriptor of a **new** socket as the return value. You then use the new socket to send and receive data.

See Also

`NetLibSocketBind()`, `NetLibSocketListen()`

NetLibSocketAddr Function

Purpose Returns the local and remote addresses currently associated with a socket.

Declared In `NetMgr.h`

Prototype

```
Int16 NetLibSocketAddr (UInt16 libRefnum,
   NetSocketRef socketRef,
   NetSocketAddrType *locAddrP,
   Int16 *locAddrLenP,
   NetSocketAddrType *remAddrP,
   Int16 *remAddrLenP,
   Int32 timeout, Err *errP)
```

Parameters

→ *`libRefNum`*
Reference number of the net library.

→ *`socketRef`*
Descriptor for the open socket.

← *`locAddrP`*
Local address of socket is returned here.

↔ *`locAddrLenP`*
On entry, length of `locAddrP` buffer in bytes. On exit, length of returned address stored in `*locAddrP`.

← *`remAddrP`*
Address of remote host is returned here.

↔ *`remAddrLenP`*
On entry, length of `remAddrP` buffer in bytes. On exit, length of returned address stored in `*remAddrP`.

→ *`timeout`*
Maximum timeout in system ticks; -1 means wait forever.

← *`errP`*
Contains an error code if the return value is -1.

Returns Returns 0 upon success and -1 if an error occurred. If the return value is -1, `errP` contains one of the following values:

`0`
No error.

`netErrTimeout`
Call timed out.

`netErrNotOpen`
The referenced net library has not been opened yet.

`netErrParamErr`

`netErrSocketNotOpen`

`netErrSocketClosedByRemote`

`netErrOutOfCmdBlocks`

Sockets Equivalent

```
int getpeername (int s, struct sockaddr *name,
int *namelen);
```

```
int getsockname (int s, struct sockaddr *name,
int *namelen);
```

Comments

This call is mainly useful for stream-based sockets. It allows the caller to find out what address was bound to a connected socket and the address of the remote host that it's connected to.

In Palm OS version 3.0 and higher, if you pass a raw socket to this function, it returns the instance number and creator of the interface to which the socket is bound.

See Also

`NetLibSocketBind()`, `NetLibSocketConnect()`, `NetLibSocketAccept()`

NetLibSocketBind Function

Purpose

Assign a local address to a socket.

Declared In

`NetMgr.h`

Prototype

```
Int16 NetLibSocketBind (UInt16 libRefnum,
   NetSocketRef socket,
   NetSocketAddrType *sockAddrP,
   Int16 addrLen, Int32 timeout, Err *errP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *socket*
Descriptor for the open socket.

→ *sockAddrP*
Pointer to the address to give to the socket. This can be a `NetSocketAddrINType` or a `NetSocketAddrRawType`.

→ *addrLen*
Length of address in `*sockAddrP`.

→ *timeout*
Maximum timeout in system ticks; -1 means wait forever.

← *errP*
Contains an error code if the return value is -1.

Returns Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

`0`
No error.

`netErrTimeout`
Call timed out.

`netErrNotOpen`
The referenced net library has not been opened yet.

`netErrParamErr`

`netErrSocketNotOpen`

`netErrSocketAlreadyConnected`

`netErrSocketClosedByRemote`

`netErrOutOfCmdBlocks`

Sockets Equivalent

```
int bind (int socket, const void *sockAddrP,
int addrLen);
```

Comments Applications that want to wait for an incoming connection request from a remote host must call this function. After calling `NetLibSocketBind`, applications can call `NetLibSocketListen()` and then `NetLibSocketAccept()` to make the socket ready to accept connection requests.

Compatibility Raw sockets are only supported in Palm OS version 3.0 and higher. See `NetLibSocketOpen()` for instructions on how to bind raw sockets.

See Also `NetLibSocketConnect()`, `NetLibSocketListen()`, `NetLibSocketAccept()`

NetLibSocketClose Function

Purpose Close a socket.

Declared In `NetMgr.h`

Prototype

```
Int16 NetLibSocketClose (UInt16 libRefnum,
   NetSocketRef socket, Int32 timeout, Err *errP)
```

Parameters

→ *`libRefNum`*
Reference number of the net library.

→ *`socket`*
Descriptor for the open socket.

→ *`timeout`*
Maximum timeout in system ticks; -1 means wait forever.

← *`errP`*
Contains an error code if the return value is -1.

Returns Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

`0`
No error.

`netErrTimeout`
Call timed out.

`netErrNotOpen`
The referenced net library has not been opened yet.

`netErrParamErr`

`netErrSocketNotOpen`

`netErrOutOfCmdBlocks`

Sockets Equivalent

```
int close (int socket);
```

Comments Closes down a socket and frees all memory associated with it.

See Also `NetLibSocketOpen()`, `NetLibSocketShutdown()`

NetLibSocketConnect Function

Purpose Assign a destination address to a socket and initiate three-way handshake if it's stream based.

Declared In `NetMgr.h`

Prototype

```
Int16 NetLibSocketConnect (UInt16 libRefnum,
   NetSocketRef socket,
   NetSocketAddrType *sockAddrP,
   Int16 addrLen, Int32 timeout, Err *errP)
```

Parameters

→ *`libRefNum`*
Reference number of the net library.

→ *`socket`*
Descriptor for the open socket.

→ *`sockAddrP`*
Pointer to address to connect to.

→ *`addrLen`*
Length of address in `*sockAddrP.`

→ *`timeout`*
Maximum timeout in system ticks; -1 means wait forever.

← *`errP`*
Contains an error code if the return value is -1.

Returns Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

0
No error.

`netErrTimeout`
Call timed out.

`netErrNotOpen`
The referenced net library has not been opened yet.

`netErrParamErr`

`netErrSocketNotOpen`

`netErrSocketBusy`

`netErrNoInterfaces`
Incorrect setup.

`netErrPortInUse`

`netErrQuietTimeNotElapsed`

`netErrInternal`

`netErrSocketAlreadyConnected`

`netErrSocketClosedByRemote`

`netErrTooManyTCPConnections`

`netErrWouldBlock`

`netErrWrongSocketType`

`netErrOutOfCmdBlocks`

Sockets Equivalent

```
int connect (int socket, const void *sockAddrP,
int addrLen);
```

See Also NetLibSocketBind(), NetUTCPOpen()

NetLibSocketListen Function

Purpose Put a stream-based socket into passive listen mode.

Declared In `NetMgr.h`

Prototype

```
Int16 NetLibSocketListen (UInt16 libRefnum,
   NetSocketRef socket, UInt16 queueLen,
   Int32 timeout, Err *errP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *socket*
Descriptor for the open socket.

→ *queueLen*
Maximum number of pending connections allowed.

→ *timeout*
Maximum timeout in system ticks; -1 means wait forever.

← *errP*
Contains an error code if the return value is -1.

Returns Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

`0`
: No error.

`netErrTimeout`
: Call timed out.

`netErrNotOpen`
: The referenced net library has not been opened yet.

`netErrParamErr`

`netErrOutOfResources`

`netErrSocketNotOpen`

`netErrSocketBusy`

`netErrNoInterfaces`
: Incorrect setup.

`netErrPortInUse`

`netErrInternal`

`netErrSocketAlreadyConnected`

`netErrSocketClosedByRemote`

`netErrWrongSocketType`

`netErrQuietTimeNotElapsed`

`netErrOutOfCmdBlocks`

Sockets Equivalent

```
int listen (int socket, int queueLen);
```

Comments

Sets the maximum allowable length of the queue for pending connections. This call is only applicable to stream-based (TCP/IP) sockets.

After a socket is created and bound to a local address using `NetLibSocketBind()`, a server application can call `NetLibSocketListen` and then `NetLibSocketAccept()` to accept connections from remote clients.

The `queueLen` is currently quietly limited to 1 (higher values are ignored).

See Also

`NetLibSocketBind()`, `NetLibSocketAccept()`

NetLibSocketOpen Function

Purpose Open a new socket.

Declared In `NetMgr.h`

Prototype

```
NetSocketRef NetLibSocketOpen (UInt16 libRefnum,
    NetSocketAddrEnum domain,
    NetSocketTypeEnum type,
    Int16 protocol, Int32 timeout, Err *errP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *domain*
Address domain. See `NetSocketAddrEnum`.

→ *type*
Desired type of connection. See `NetSocketTypeEnum`.

→ *protocol*
Protocol to use. This parameter is currently ignored.

For raw sockets in the `netSocketAddrINET` domain, specify one of the following:

`netSocketProtoIPTCP`

`netSocketProtoIPUDP`

`netSocketProtoIPRAW`
For all other socket types or for raw sockets in the raw domain, this parameter is ignored.

→ *timeout*
Maximum timeout in system ticks; -1 means wait forever.

← *errP*
Contains an error code if the return value is -1.

Returns Returns the `NetSocketRef` of the opened socket or -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

`0`
No error.

`netErrTimeout`
Call timed out.

`netErrNotOpen`
: The referenced net library has not been opened yet.

`netErrParamErr`

`netErrNoMoreSockets`

`netErrOutOfCmdBlocks`

`netErrOutOfMemory`

Sockets Equivalent

```
int socket (int domain, int type, int protocol);
```

Comments

Allocates memory for a new socket and opens it.

Raw sockets are supported in Palm OS version 3.0 and higher. Two types of raw sockets are supported:

- Raw sockets in the `netSocketAddrINET` domain

  In this case, you must bind the socket to an IP address using `NetLibSocketBind()`, passing a `NetSocketAddrINType` structure for the socket address. The port field is ignored.

  For applications that use raw sockets in the INET domain, the net library checks the destination IP address of all incoming packets to see if it matches any of those raw sockets. If it does, the packet is enqueued directly into the matching socket and is *not* passed to the protocol stack.

  When an application sends data through raw sockets in the IP domain, the net library packages the data into a packet and passes it directly to the interface's send routine. You are responsible for forming the entire IP header, including any necessary checksums, source and destination IP address, and so on.

- Raw sockets in the `netSocketAddrRaw` domain with no protocol

  In this case, you must bind the socket to an interface using `NetLibSocketBind()`, passing a `NetSocketAddrRawType` structure for the socket address. The instance and creator specify which interface the caller wants to receive raw packets from.

  When an interface is bound to a raw socket with no protocol, the net library places that interface into raw mode. In raw

mode, the interface passes all incoming packets, no matter what the link layer protocol, to its raw receive function.

When an application sends data through a raw socket with no protocol, the net library packages the data into a packet and passes it directly to the interface's send routine.

The interface remains in raw mode until the raw socket is closed.

Compatibility Raw sockets supported only in Palm OS version 3.0 and higher.

See Also `NetLibSocketClose()`, `NetUTCPOpen()`

NetLibSocketOptionGet Function

Purpose Retrieves the current value of a socket option.

Declared In `NetMgr.h`

Prototype

```
Int16 NetLibSocketOptionGet (UInt16 libRefnum,
   NetSocketRef socket, UInt16 level,
   UInt16 option,
   void *optValueP, UInt16 *optValueLenP,
   Int32 timeout, Err *errP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *socket*
Descriptor for the open socket.

→ *level*
Level of the option, one of the `NetSocketOptLevelEnum` constants. See `NetLibSocketOptionSet()`.

→ *option*
One of the `NetSocketOptEnum` constants. See `NetLibSocketOptionSet()`.

← *optValueP*
Pointer to variable holding new value of option.

↔ *optValueLenP*
Size of variable pointed to by `optValueP` on entry. Actual size of return value on exit.

→ *timeout*
Maximum timeout in system ticks; -1 means wait forever.

← *errP*
Contains an error code if the return value is -1.

Returns Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

`0`
No error.

`netErrTimeout`
Call timed out.

`netErrNotOpen`
The referenced net library has not been opened yet.

`netErrParamErr`

`netErrSocketNotOpen`

`netErrUnimplemented`

`netErrWrongSocketType`

`netErrInvalidSettingSize`

Sockets Equivalent

```
int getsockopt (int socket, int level, int option,
const void *optValueP, int *optValueLenP);
```

Comments Returns the current value of a socket option. The caller passes a pointer to a variable to hold the returned value (in `optValueP`) and the size of this variable (in `*optValueLenP`). On exit, `*optValueP` is updated with the actual size of the return value.

For all of the fixed size options (every option except `netSocketOptIPOptions`), `*optValueLenP` is unmodified on exit and this call does its best to return the value in the caller's desired type size.

For compatibility with existing Internet applications, this call is quite flexible on the `*optValueLenP` parameter. If the desired type for an option is `FLAG`, this call supports an `*optValueLenP` of 1, 2, or 4. If the desired type for an option is `int`, it supports an `*optValueLenP` of 2 or 4.

See NetLibSocketOptionSet() for a list of available options.

See Also NetLibSocketOptionSet()

NetLibSocketOptionSet Function

Purpose Set a socket option.

Declared In NetMgr.h

Prototype

```
Int16 NetLibSocketOptionSet (UInt16 libRefnum,
   NetSocketRef socket, UInt16 level,
   UInt16 option,
   void *optValueP, UInt16 optValueLen,
   Int32 timeout, Err *errP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *socket*
Descriptor for the open socket.

→ *level*
Level of the option, one of the NetSocketOptLevelEnum constants. See the comments section.

→ *option*
One of the NetSocketOptEnum constants. See the comments section.

→ *optValueP*
Pointer to the variable holding the new value of the option.

→ *optValueLen*
Size of variable pointed to by optValueP.

→ *timeout*
Maximum timeout in system ticks; -1 means wait forever.

← *errP*
Contains an error code if the return value is -1.

Returns Returns 0 upon success and -1 if an error occurred. If an error occurred, errP contains one of the following values:

0
No error.

`netErrTimeout`
: Call timed out.

`netErrNotOpen`
: The referenced net library has not been opened yet.

`netErrParamErr`

`netErrSocketNotOpen`

`netErrUnimplemented`

`netErrWrongSocketType`

`netErrInvalidSettingSize`

Sockets Equivalent

```
int setsockopt (int socketRef, int level,
int option, const void *optValueP,
int optValueLen);
```

Comments

Sets various options associated with a socket. The caller passes a pointer to the new option value in `optValueP` and the size of the option in `optValueLen`.

Table 66.4 lists the available options.

- The Level column specifies the option level, which is one of the `netSocketOptLevelXXX` constants.
- The Option column lists the option, which is one of the `netSocketOptXXX` constants.
- The G/S column lists whether this option can be fetched with the `NetLibSocketOptionGet()` call (G) and/or set (S) with this call.
- The type column lists the data type of the option.
- The I column specifies whether or not this option is currently implemented.

Table 66.4 Net Library Socket Options

netSocket OptLevel...	netSocketOpt...	G/S	Type	I	Description
IP	IPOptions	GS	`UInt8[]`	N	Options in IP Header
TCP	TCPNoDelay	GS	`FLAG`	Y	Don't delay send to coalesce packets
TCP	TCPMaxSeg	G	`int`	Y	Get TCP maximum segment size
Socket	SockDebug	GS	`FLAG`	N	Turn on recording of debug info
Socket	SockAcceptConn	G	`FLAG`	N	Socket has had listen
Socket	SockReuseAddr	GS	`FLAG`	N	Allow local address reuse
Socket	SockKeepAlive	GS	`FLAG`	Y	Keep connections alive
Socket	SockDontRoute	GS	`FLAG`	N	Just use interface addresses
Socket	SockBroadcast	GS	`FLAG`	N	Permit sending of broadcast messages
Socket	SockUseLoopback	GS	`FLAG`	N	Bypass hardware when possible
Socket	SockLinger	GS	`NetSocketLingerType`	Y	Linger on close if data present `NetSocketLingerType` is a structure with two fields: `onOff` (`true` or `false`) and `time` (linger time in seconds).
Socket	SockOOBInLine	GS	`FLAG`	N	Leave received OOB data in-line
Socket	SockSndBufSize	GS	`int`	N	Send buffer size

Table 66.4 Net Library Socket Options *(continued)*

netSocket OptLevel...	netSocketOpt...	G/S	Type	I	Description
Socket	SockRcvBufSize	GS	`int`	N	Receive buffer size
Socket	SockSndLowWater	GS	`int`	N	Send low-water mark
Socket	SockRcvLowWater	GS	`int`	N	Receive low-water mark
Socket	SockSndTimeout	GS	`int`	N	Send timeout
Socket	SockRcvTimeout	GS	`int`	N	Receive timeout
Socket	SockErrorStatus	G	`int`	Y	Get error status and clear
Socket	SockSocketType	G	`int`	Y	Get socket type
Socket	SockNonBlocking	GS	`FLAG`	Y	Set non-blocking mode on/off
Socket	SockNotice	S	`NetSocketNoticeType`	Y	Enable socket notices. Your application will receive a notice when certain socket conditions occur. Specify conditions you want to watch in the `optValueP->conditions` field.
Socket	SockRequireErrClear	GS	FLAG	Y	Return the current error status for all subsequent socket function calls until the error is cleared.

For compatibility with existing Internet applications, this call is quite flexible on the `optValueLen` parameter. If the desired type for an option is `FLAG`, this call accepts an `optValueLen` of 1, 2, or 4. If the desired type for an option is `int`, it accepts an `optValueLen` of 2 or 4.

Except for the `netSocketOptSockNonBlocking` option, all options listed above have equivalents in the sockets API. The `netSocketOptSockNonBlocking` option was added to this call in the net library in order to implement the functionality of the UNIX `fcntl()` control call, which can be used to turn nonblocking mode on and off for sockets.

See Also `NetLibSocketOptionGet()`

NetLibSocketShutdown Function

Purpose Shut down a socket in one or both directions.

Declared In `NetMgr.h`

Prototype

```
Int16 NetLibSocketShutdown (UInt16 libRefnum,
   NetSocketRef socket, Int16 direction,
   Int32 timeout, Err *errP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *socket*
Descriptor for the open socket.

→ *direction*
Direction to shut down. One of the `NetSocketDirEnum` constants. Specifically:

`netSocketDirInput`

`netSocketDirOutput`

`netSocketDirBoth`

→ *timeout*
Maximum timeout in system ticks; -1 means wait forever.

← *errP*
Contains an error code if the return value is -1.

Returns Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

0
No error.

`netErrTimeout`
: Call timed out.

`netErrNotOpen`
: The referenced net library has not been opened yet.

`netErrParamErr`

`netErrSocketNotOpen`

`netErrNoMultiPktAddr`

`netErrOutOfCmdBlocks`

Sockets Equivalent

```
int shutdown (int socket, int direction);
```

Comments

Shuts down communication in one or both directions on a socket.

If direction is `netSocketDirInput`, the socket is marked as down in the receive direction and further read operations from it return a `netErrSocketInputShutdown` error.

NetLibTracePrintF Function

Purpose

Store debugging information in the net library's trace buffer.

Declared In

`NetMgr.h`

Prototype

```
Err NetLibTracePrintF (UInt16 libRefNum,
   const Char *formatStr, ...)
```

Parameters

→ *`libRefNum`*
: Reference number of the net library.

→ *`formatStr`*
: A `printf` style format string.

→ `...`
: Arguments to the format string.

Returns

Returns 0 upon success or `netErrNotOpen` if the net library has not been opened.

Sockets Equivalent

None

Comments

This call is a convenient debugging tool for developing Internet applications. It stores a message into the net library's trace buffer,

which can later be dumped using the `NetLibMaster()` call. The net library's trace buffer is used to store run-time errors that the net library encounters as well as errors and messages from network interfaces and from applications that use this call.

The `formatStr` parameter is a `printf` style format string which supports the following format specifiers:

%d, %i, %u, %x, %s, %c

but it does *not* support field widths, leading 0's etc.

Note that the `netTracingAppMsgs` bit of the `netSettingTraceBits` setting must be set using the call `NetLibSettingSet(...netSettingTraceBits...)`. Otherwise, this routine will do nothing.

See Also `NetLibTracePutS()`, `NetLibMaster()`, `NetLibSettingSet()`

NetLibTracePutS Function

Purpose Store debugging information in the net library's trace buffer.

Declared In `NetMgr.h`

Prototype

```
Err NetLibTracePutS (UInt16 libRefNum,
    Char *strP)
```

Parameters

→ *libRefNum*
Reference number of the net library.

→ *strP*
String to store in the trace buffer.

Returns Returns 0 upon success or `netErrNotOpen` if the net library has not been opened.

Sockets Equivalent None

Comments This call is a convenient debugging tool for developing Internet applications. It will store a message into the net library's trace buffer which can later be dumped using the `NetLibMaster()` call. The net library's trace buffer is used to store run-time errors that the net library encounters as well as errors and messages from network interfaces and from applications that use this call.

Note the `netTracingAppMsgs` bit of the `netSettingTraceBits` setting must be set using the `NetLibSettingSet(...netSettingTraceBits...)` call or this routine will do nothing.

See Also NetLibTracePrintF(), NetLibMaster(), NetLibSettingSet()

NetNToHL Macro

Purpose Converts a 32-bit value from network to host byte order.

Declared In `NetBitUtils.h`

Prototype

```
#define NetNToHL (x)
```

Parameters → *x*
32-bit value to convert.

Returns Returns `x` in host byte order.

Errors none

Sockets Equivalent `ntohl()`

See Also `NetNToHS()`, `NetHToNL()`, `NetHToNS()`

NetNToHS Macro

Purpose Converts a 16-bit value from network to host byte order.

Declared In `NetBitUtils.h`

Prototype

```
#define NetNToHS (x)
```

Parameters → *x*
16-bit value to convert.

Returns Returns `x` in host byte order.

Errors None

Sockets Equivalent `ntohs()`

See Also NetHToNL(), NetNToHL(), NetHToNS()

67

Network Utilities

This chapter describes network utilities provided in the module `NetSocket.c`. These utilities are convenience functions that you can use in place of net library functions in applications that use the net library. You can find `NetSocket.c` in the folder `Libraries\Net\Src`. (On Palm OS® 3.5, `NetSocket.c` is in the folder `CodeWarrior Libraries\Comms\NetSocket\Src`.)

The include file for the functions described in this chapter is `<unix/sys_socket.h>`. This header file is not included by any other Palm™ header file; you must explicitly include it in your code.

For more information on `NetSocket.c` and `sys_socket.h`, see the chapter "Network Communication" in the *Palm OS Programmer's Companion*, vol. II, *Communications*.

Network Utility Functions

NetUReadN Function

Purpose Reads a specified number of bytes from a socket.

Declared In `unix/sys_socket.h`

Prototype

```
Int32 NetUReadN (NetSocketRef fd, UInt8 *bufP,
    UInt32 numBytes)
```

Parameters

→ *fd*
Descriptor for the open socket.

← *bufP*
Pointer to buffer to hold received data.

→ *numBytes*
Number of bytes to read.

Returns Returns the number of bytes actually read. If the return value is less than 0, an error occurred.

Comments This function repeatedly calls `NetLibReceive()` until `numBytes` have been read or until `NetLibReceive` returns an error.

See Also `NetUWriteN()`

NetUTCPOpen Function

Purpose Opens a TCP (streams-based) socket and connects it to a server.

Declared In `unix/sys_socket.h`

Prototype

```
NetSocketRef NetUTCPOpen (Char *hostName,
    Char *serviceName, Int16 port)
```

Parameters

→ *hostName*
Remote host, given either by name or by dotted decimal address.

→ *serviceName*
The name of a network service. Possible services are "echo", "discard", "daytime", "qotd", "chargen", "ftp-data", "ftp", "telnet", "smtp", "time", "name", "finger", "pop2", "pop3", "nntp", "imap2". The value of this parameter is ignored if the `port` parameter is greater than zero.

→ *port*
The number of the port to connect to on the remote host. Set `port` to zero to use `serviceName` instead.

Returns Returns the socket descriptor of the socket that was connected, or -1 if an error occurred.

Comments If `serviceName` is given and `port` is less than or equal to zero, this function looks up the port number for that service on the remote host and uses it for the connection.

This function is the equivalent of calling `NetLibSocketOpen()` and `NetLibSocketConnect()` (or `socket` and `connect`).

NOTE: This function does not return specific reasons for failure if there is a failure. This function is not production-quality code. It is provided as a quick and dirty way of creating a connection and as sample code that can be used as a reference.

NetUWriteN Function

Purpose Writes the specified number of bytes to a socket.

Declared In `unix/sys_socket.h`

Prototype

```
Int32 NetUWriteN (NetSocketRef fd, UInt8 *bufP,
   UInt32 numBytes)
```

Parameters

→ *fd*
Descriptor for the open socket.

→ *bufP*
Pointer to buffer to write.

→ *numBytes*
Number of bytes to write.

Returns Returns the number of bytes actually sent. If the return value is less than 0, an error occurred.

Comments This function repeatedly calls `NetLibSend()` until `numBytes` have been written or until `NetLibSend` returns an error.

See Also `NetUReadN`

68

Script Plugin

This chapter describes the login script plugin support. You write a plugin to add to the list of available login script commands in the Network preferences panel. This chapter covers:

- Script Plugin Data Types
- Script Plugin Constants

The header file `ScriptPlugin.h` declares the API described in this chapter.

For more information on the script plugin, see the section "Extending the Network Login Script Support" on page 184 in the "Network Communication" chapter of the *Palm OS Programmer's Companion*, vol. II, *Communications*.

Script Plugin Data Types

PluginCallbackProcType Struct

Purpose The `PluginCallbackProcType` defines the `procP` field in `PluginExecCmdType`.

Prototype

```
typedef struct {
  ScriptPluginSelectorProcPtr selectorProcP;
} PluginCallbackProcType, *PluginCallbackProcPtr;
```

Fields

`selectorProcP`
: The address of a selector-based callback function for accessing the functionality of the network interface. Each network interface provides it own `ScriptPluginSelectorProc` function. See `ScriptPluginSelectorProc()`.

PluginCmdPtr Typedef

Purpose The `PluginCmdPtr` type defines a pointer to a `PluginCmdType` structure.

Prototype

```
typedef PluginCmdType *PluginCmdPtr;
```

PluginCmdType Struct

Purpose The `PluginCmdType` structure specifies the name of a command.

Prototype

```
typedef struct {
  Char commandName[pluginMaxCmdNameLen + 1];
  Boolean hasTxtStringArg;
  UInt8 reserved;
} PluginCmdType;
```

Fields

`commandName`
: The name of the command. This string appears in the pull-down list in the Network preferences panel's script view.

  The pull-down list contains all available commands from all plugins. Make sure that your command name is unique and as short as possible.

`hasTxtStringArg`
: `true` if the command takes an argument. In this case when the user selects this command, the Network preferences panel displays a field next to the command name where the user should enter the argument. This argument is passed in the `txtStringArg` field in `PluginExecCmdType` when the command is to be executed.

`reserved`
: Reserved for future use.

PluginExecCmdType Struct

Purpose The `PluginExecCmdType` structure defines the parameter block for the `scptLaunchCmdExecuteCmd` launch code. This structure specifies which command is to be executed and provides any necessary arguments for the command. Your plugin should respond by executing the command.

Prototype

```
typedef struct {
  Char commandName[pluginMaxCmdNameLen + 1];
  Char txtStringArg [pluginMaxLenTxtStringArg + 1];
  PluginCallbackProcPtr procP;
  void *handle;
} PluginExecCmdType, *PluginExecCmdPtr;
```

Fields

`commandName`
: The command's name. This is the string that appears in the pull-down list in the script view of the Network preferences panel.

`txtStringArg`
: If the command takes an argument, this field provides the argument as a string. A `NULL` value means either that the user did not provide a value, or that you didn't specify that the command takes an argument.

`procP`
: A pointer to a `PluginCallbackProcType` structure, which identifies the network interface function that the plugin can use to execute the command.

`handle`
: Handle to information specific to a particular connection. You must pass this value when you call the function pointed to by `procP`.

PluginInfoPtr Typedef

Purpose The `PluginInfoPtr` type defines a pointer to a `PluginInfoType` structure.

Prototype

```
typedef PluginInfoType *PluginInfoPtr;
```

PluginInfoType Struct

Purpose The `PluginInfoType` structure is the parameter block for the `scptLaunchCmdListCmds` launch code. When your plugin receives the launch code, the `PluginInfoType` structure is empty. Your plugin should fill in the `PluginInfoType` and return it. The system uses the information returned to construct the pull-down list

of available script commands and build a table of which plugin will execute which script command.

Prototype

```
typedef struct {
  Char  pluginName[pluginMaxModuleNameLen + 1];
  UInt16  numOfCommands;
  PluginCmdType  command[pluginMaxNumOfCmds];
} PluginInfoType;
```

Fields

`pluginName`
: A name that the system can use to identify your plugin. This is typically the same name you give the PRC file.

`numOfCommands`
: The number of commands that your plugin defines. The maximum allowed is `pluginMaxNumOfCmds`.

`command`
: An array of `PluginCmdType` structures that provide information about the commands that your plugin defines.

ScriptPluginLaunchCodesEnum Enum

Purpose

The `ScriptPluginLaunchCodesEnum` defines the launch codes for the script plugin. Your script plugin's `PilotMain()` function should respond to the launch codes defined in this enum.

Prototype

```
typedef enum {
  scptLaunchCmdDoNothing =
      sysAppLaunchCmdCustomBase,
  scptLaunchCmdListCmds,
  scptLaunchCmdExecuteCmd
} ScriptPluginLaunchCodesEnum;
```

Constants

`scptLaunchCmdDoNothing`
: This launch code is a no-op supplied only to provide a beginning value for the script plugin launch codes. It is not necessary to respond to this launch code.

`scptLaunchCmdListCmds`
: Provide information about the commands that your plugin executes. See `PluginInfoType`.

`scptLaunchCmdExecuteCmd`
: Execute the specified command.

This launch code is received when the system is executing a user's login script during a network connection attempt. Your plugin should respond by executing the command provided in the `PluginExecCmdType` parameter block.

Script Plugin Constants

Command Constants

The following constants identify the available commands that the network interface can perform for you. These commands are building blocks that you use to create your own script commands. To perform one of these tasks, pass the constant value as an argument to the network interface's callback function (`ScriptPluginSelectorProc()`).

Constant	Value	Description
`pluginNetLibDoNothing`	0	For debugging purposes.
`pluginNetLibReadBytes`	1	Read the specified number of bytes from the open connection.
`pluginNetLibWriteBytes`	2	Write the specified number of bytes to the open connection.
`pluginNetLibGetUserName`	3	Get the user name from the network service profile.
`pluginNetLibGetUserPwd`	4	Get the user's password from the network service profile.
`pluginNetLibCheckCancelStatus`	5	Check to see if the user canceled the connection.
`pluginNetLibPromptUser`	6	Prompt the user for input.
`pluginNetLibConnLog`	7	Write a string to the network service's connection log.

Constant	Value	Description
`pluginNetLibCallUIProc`	8	Have the network interface call a function in your plugin that displays UI. Use this command if you need to display a more complicated user interface than the simple user prompt that the network interface provides.
`pluginNetLibGetSerLibRefNum`	9	Obtain the serial library's reference number. You need the reference number to perform any serial library commands, which you might need to perform more complex work with the connection port.

Size Constants

The following table lists constants that control the size of strings in your plugin and the size of the plugin itself.

Constant	Value	Description
`pluginMaxCmdNameLen`	15	The maximum length for the command's name, not including the terminating null character. This is the string displayed to the user in the pull-down menu.
`pluginMaxModuleNameLen`	15	The maximum length for the plugin's name (not including the terminating null character), which is typically the name of the PRC file as well.

Constant	Value	Description
`pluginMaxNumOfCmds`	10	The maximum number of commands that your plugin can define.
`pluginMaxLenTxtStringArg`	63	The maximum length of the argument that each command can take, not including the terminating null character.

Script Plugin Functions

ScriptPluginSelectorProc Function

Purpose A function provided by the network interface for the purpose of performing script commands.

Declared In `ScriptPlugin.h`

Prototype

```
Err (*ScriptPluginSelectorProcPtr) (void *handle,
   UInt16 command, void *dataBufferP,
   UInt16 *sizeP, UInt16 *dataTimeoutP,
   void *procAddrP)
```

Parameters

→ *handle*
Handle to information specific to a particular connection.

→ *command*
The command to be executed. See "Command Constants" for a list of possible values. The rest of the parameters to this callback function are interpreted differently based on the value of the `command` parameter. See the table in the "Comments" section for specifics.

↔ *dataBufferP*
A pointer to arguments to pass to the command or a pointer to data returned by the command. See the "Comments" section below.

↔ *sizeP*
The size of `dataBufferP`.

→ *dataTimeoutP*
Number of seconds to wait for the command to execute. 0 means wait forever. Applies only to commands that request information from the network.

→ *procAddrP*
Pointer to a user interface callback function that the network interface should call to complete the function. Used only by `pluginNetLibCallUIProc`. This function should take one argument of the same type that you pass to `dataBufferP` and should return `void`.

Returns Returns 0 upon success, or an error condition upon failure. If an error condition is returned, your plugin should stop processing and return the error condition from its `PilotMain()`.

Comments When your plugin receives the `scptLaunchCmdExecuteCmd` launch code, the parameter block contains the command's name, its text string argument (if any), and a pointer to the network interface's callback function. You should use this callback function any time you need to communicate with the network library, the user, or the host computer during execution of your command.

The callback function takes as arguments the handle to information about this connection (which is also passed in the launch code's parameter block), and the command that the service should execute. The rest of the parameters are interpreted differently based on what the value the command argument is. See the table below.

pluginNetLib	dataBufferP	sizeP	dataTimeOutP	procAddrP
`DoNothing`	N/A	N/A	N/A	N/A
`ReadBytes`	On return, contains the bytes that were read.	On input, contains the number of bytes to read. On return, contains the number of bytes actually read.	Number of seconds to wait before timing out the operation.	N/A

pluginNetLib	dataBufferP	sizeP	dataTimeOutP	procAddrP
`WriteBytes`	On input, contains the data to send.	On input, contains the number of bytes to send. On return, contains the number of bytes actually sent.	Number of seconds to wait for a response before canceling.	N/A
`UserName`	On return, contains the user's name	On return, contains the size of the string pointed to by `dataBufferP`.	N/A	N/A
`UserPwd`	On return, contains the user's password.	On return, contains the size of the string pointed to by `dataBufferP`.	N/A	N/A
`CheckCancelStatus`	On return, the Boolean value `true` if the user canceled the command, `false` otherwise.	Size of Boolean.	N/A	N/A
`PromptUser`	On input, the prompt to display. On return, the text that the user entered.	On input and on return, the size of the string pointed to by `dataBufferP`.	N/A	N/A

pluginNetLib	dataBufferP	sizeP	dataTimeOutP	procAddrP
`ConnLog`	The string that should be written to the log.	N/A	N/A	N/A
`CallUIProc`	A pointer to a structure to pass to your callback function as a parameter. This structure should contain a handle to the form to be displayed, plus any other necessary information.	N/A	N/A	A pointer to a function in your plugin that displays the form.
`GetSerLib RefNum`	On return, contains the serial library's reference number.	N/A	N/A	N/A

69

Virtual Drivers

This chapter provides reference material for the Serial Manager virtual device driver API:

- Driver Data Structures
- Driver Constants
- Virtual Driver-Defined Functions
- Serial Manager Queue Functions

The header files `SerialVdrv.h` and `SerialDrvr.h` declare the virtual driver API. For more information on writing device drivers for the Serial Manager, see section "Writing a Virtual Device Driver" on page 114 in the "Serial Communication" chapter of *Palm OS Programmer's Companion*, vol. II, *Communications*.

Driver Data Structures

DrvrInfoType Struct

Purpose The `DrvrInfoType` structure defines information about the serial hardware. It is passed to and filled in by the `DrvEntryPointProcPtr()` for a virtual driver.

Prototype

```
typedef struct {
  UInt32 drvrID;
  UInt32 drvrVersion;
  UInt32 maxBaudRate;
  UInt32 handshakeThreshold;
  UInt32 portFlags;
  const Char *portDesc;
  DrvrIRQEnum irqType;
  UInt8 multipleEnumerations;
  UInt32 dbCreator;
} DrvrInfoType;
```

Fields

`drvrID`
4-character creator type, such as 'u328'.

`drvrVersion`
Version of code that works for this hardware. See Driver Version Constants.

`maxBaudRate`
Maximum baud rate supported by this hardware.

`handshakeThreshold`
Baud rate at which the use of hardware handshaking is necessary.

`portFlags`
Bit flags denoting features of this hardware. The flags are described in Port Feature Constants.

`portDesc`
Pointer to null-terminated string describing this hardware. This string appears in the Connection panel to describe the port to the user (only if the `portCncMgrVisible` bit in `portFlags` is set). Can be `NULL` if the driver contains a resource (of type 'tSTR' and id `kPortDescStrID`) that supplies this string.

`irqType`
IRQ line being used for this hardware. For a virtual driver, specify `drvrIRQNone`.

`multipleEnumerations`
The number of entries in the driver table required for this driver. If 0, the driver has a single entry.

`dbCreator`
Creator ID of the database containing this driver.

Compatibility

The `multipleEnumerations` and `dbCreator` fields are only defined if New Serial Manager Feature Set Version 2 is present.

DrvrRcvQType Struct

Purpose

The `DrvrRcvQType` structure defines the virtual driver receive buffer and function pointers to functions that access and save data to the buffer. A pointer to this structure is passed to the

VdrvOpenProcPtr() function. The DrvrHWRcvQPtr type defines a pointer to a DrvrRcvQType structure.

Prototype

```
typedef struct DrvrRcvQType {
  void *rcvQ;
  WriteByteProcPtr qWriteByte;
  WriteBlockProcPtr qWriteBlock;
  GetSizeProcPtr qGetSize;
  GetSpaceProcPtr qGetSpace;
  SignalCheckPtr qSignalCheck;
} DrvrRcvQType;
typedef DrvrRcvQType *DrvrHWRcvQPtr;
```

Fields

`rcvQ`
: Pointer to the receive buffer.

`qWriteByte`
: Function pointer to a function that the virtual driver can use to write one byte to the Serial Manager's receive queue. See the WriteByteProcPtr() function.

`qWriteBlock`
: Function pointer to a function that the virtual driver can use to write a block of bytes to the Serial Manager's receive queue. See the WriteBlockProcPtr() function.

`qGetSize`
: Function pointer to a function that the virtual driver can use to get the total size of the Serial Manager's receive queue. See the GetSizeProcPtr() function.

`qGetSpace`
: Function pointer to a function that the virtual driver can use to get the available space in the Serial Manager's receive queue. See the GetSpaceProcPtr() function.

`qSignalCheck`
: Function pointer to a function that the virtual driver can use to perform a signal check for the Serial Manager's receive queue. See the SignalCheckPtr() function.

Compatibility

The qSignalCheck field is only defined if New Serial Manager Feature Set Version 2 is present.

DrvrStatusEnum Enum

Purpose The `DrvrStatusEnum` enumerated type specifies serial status bit flags. Return these enumerated types from the `VdrvStatusProcPtr()` call.

Prototype

```
typedef enum DrvrStatusEnum {
  drvrStatusCtsOn = 0x0001,
  drvrStatusRtsOn = 0x0002,
  drvrStatusDsrOn = 0x0004,
  drvrStatusTxFifoFull = 0x0008,
  drvrStatusTxFifoEmpty = 0x0010,
  drvrStatusBreakAsserted = 0x0020,
  drvrStatusDataReady = 0x0040,
  drvrStatusLineErr = 0x0080
} DrvrStatusEnum;
```

Constants

`drvrStatusCtsOn`
: Set if CTS line is active.

`drvrStatusRtsOn`
: Set if RTS line is active.

`drvrStatusDsrOn`
: Set if DSR is on.

`drvrStatusTxFifoFull`
: Set if transmit FIFO is full; cleared if FIFO has space.

`drvrStatusTxFifoEmpty`
: Set if transmit FIFO is empty.

`drvrStatusBreakAsserted`
: Set if sending break characters is enabled.

`drvrStatusDataReady`
: Used by debugger only.

`drvrStatusLineErr`
: Used by debugger only.

SrmRcvQType Struct

Purpose The `SrmRcvQType` structure defines the Serial Manager receive queue. This queue is passed as a parameter to the virtual driver.

Prototype

```
typedef struct SrmRcvQType {
  UInt32 qStart;
  UInt32 qEnd;
  UInt32 qSize;
  UInt8 *qData;
  void *qPort;
} SrmRcvQType;
```

Fields

`qStart`
The start of the queue.

`qEnd`
The end of the queue.

`qSize`
The size of the queue.

`qData`
The data currently in the queue.

`qPort`
A pointer to the current foreground port.

Compatibility

The `SrmRcvQType` structure was previously a private structure. It is declared publicly if New Serial Manager Feature Set Version 2 is present.

VdrvAPIType Struct

Purpose

The `VdrvAPIType` structure defines function pointers to the required virtual driver functions. When passed a pointer to this structure in the `DrvEntryPointProcPtr()` function, that function must fill in the pointers to the virtual driver functions appropriately.

Prototype

```
typedef struct {
  VdrvOpenProcPtr drvOpen;
  VdrvCloseProcPtr drvClose;
  VdrvControlProcPtr drvControl;
  VdrvStatusProcPtr drvStatus;
  VdrvReadProcPtr drvRead;
  VdrvWriteProcPtr drvWrite;
  VdrvOpenProcV4Ptr drvOpenV4;
  VdrvControlCustomProcPtr drvControlCustom;
} VdrvAPIType;
```

Fields

`drvOpen`
Pointer to the driver open function.

`drvClose`
Pointer to the driver close function.

`drvControl`
Pointer to the driver control function.

`drvStatus`
Pointer to the driver status function.

`drvRead`
Pointer to the driver read function.

`drvWrite`
Pointer to the driver write function.

`drvOpenV4`
Pointer to the driver open function for New Serial Manager Feature Set Version 2.

`drvControlCustom`
Pointer to the driver custom control function.

Compatibility

`drvOpenV4` and `drvControlCustom` are declared if both New Serial Manager Feature Set Version 2 and 4.0 New Feature Set are present.

VdrvConfigType Struct

Purpose

The `VdrvConfigType` structure specifies parameters for opening a serial port. This structure is passed as a parameter to `VdrvOpenProcV4Ptr()`.

Prototype

```
typedef struct VdrvConfigType {
  UInt32 baud;
  UInt32 drvrId;
  UInt32 function;
  MemPtr drvrDataP;
  UInt16 drvrDataSize;
  UInt32 sysReserved1;
  UInt32 sysReserved2;
} VdrvConfigType;
```

Fields

`baud`
Baud rate at which to open the connection. Serial drivers that do not require baud rates ignore this field.

`drvrId`
Creator ID of the application or library that is using the Serial Manager.

`function`
The reason why the port was opened. Specify the creator ID of the application that is opening the port or one of the following values:

`serFncUndefined`
Undefined function. This is the default value for this field.

`serFncPPPSession`
The connection is to be used for the PPP interface.

`serFncSLIPSession`
The connection is to be used for the SLIP session.

`serFncDebugger`
The connection is to be used for a debugging session.

`serFncHotSync`
The connection is to be used for a HotSync operation.

`serFncConsole`
The connection is to the debugging console.

`serFncTelephony`
The connection is to the telephony library.

The `function` field is used by protocols such as USB and Bluetooth that perform different setup tasks based on which type of application is using them. RS-232 drivers ignore this parameter.

`drvrDataP`
Pointer to a driver-specific data block.

`drvrDataSize`
The size of the data block pointed to by `drvrDataP`.

`sysReserved1`
Reserved for future use.

`sysReserved2`
Reserved for future use.

Compatibility This structure is only defined if both New Serial Manager Feature Set Version 2 and 4.0 New Feature Set are present.

VdrvCtlOpCodeEnum Enum

Purpose The `VdrvCtlOpCodeEnum` enumerated type specifies a serial control operation. You should handle each of these constants when passed for the `controlCode` parameter to the `VdrvControlProcPtr()` call.

Prototype

```
typedef enum VdrvCtlOpCodeEnum {
  vdrvOpCodeNoOp = 0,
  vdrvOpCodeSetBaudRate = 0x1000,
  vdrvOpCodeSetSettingsFlags,
  vdrvOpCodeSetCtsTimeout,
  vdrvOpCodeClearErr,
  vdrvOpCodeSetSleepMode,
  vdrvOpCodeSetWakeupMode,
  vdrvOpCodeFIFOCount,
  vdrvOpCodeStartBreak,
  vdrvOpCodeStopBreak,
  vdrvOpCodeStartLoopback,
  vdrvOpCodeStopLoopback,
  vdrvOpCodeFlushTxFIFO,
  vdrvOpCodeFlushRxFIFO,
  vdrvOpCodeSendBufferedData,
  vdrvOpCodeRcvCheckIdle,
  vdrvOpCodeEmuSetBlockingHook,
  vdrvOpCodeGetOptTransmitSize,
  vdrvOpCodeGetMaxRcvBlockSize,
  vdrvOpCodeNotifyBytesReadFromQ,
  vdrvOpCodeSetDTRAsserted,
  vdrvOpCodeGetDTRAsserted,
  vdrvOpCodeWaitForConfiguration,
  vdrvOpCodeGetUSBDeviceDescriptor,
  vdrvOpCodeGetUSBConfigDescriptor,
  vdrvOpCodeEnableIRDA,
  vdrvOpCodeDisableIRDA,
  vdrvOpCodeEnableUART,
  vdrvOpCodeDisableUART,
```

```
  vdrvOpCodeRxEnable,
  vdrvOpCodeRxDisable,
  vdrvOpCodeLineEnable,
  vdrvOpCodeEnableUARTInterrupts,
  vdrvOpCodeDisableUARTInterrupts,
  vdrvOpCodeSetReceiveQueue,
  vdrvOpCodeSaveState,
  vdrvOpCodeRestoreState,
  vdrvOpCodeSetYieldPortCallback,
  vdrvOpCodesetYieldPortRefCon,
  vdrvOpCodeUserDef = 0x2000,
  vdrvOpCodeSystem = 0x7000,
  vdrvOpCodeCustom = 0x8000
} VdrvCtlOpCodeEnum;
```

Constants

`vdvrOpCodeSetBaudRate`
: Sets the baud rate.

`vdvrOpCodeSetSettingsFlags`
: Sets the data transmission options. The bit flags are described in Serial Settings Constants.

`vdrvOpCodeSetCtsTimeout`
: Hardware handshake timeout.

`vdvrOpCodeClearErr`
: Clears the hardware error state.

`vdvrOpCodeSetSleepMode`
: Puts the port in sleep mode (not typically used for virtual drivers).

`vdvrOpCodeSetWakeupMode`
: Wakes up the port from sleep mode (not typically used for virtual drivers).

`vdvrOpCodeFIFOCount`
: Returns the number of bytes currently in the FIFO (or best estimate).

`vdvrOpCodeStartBreak`
: Sends a break character or enables the sending of break characters.

`vdvrOpCodeStopBreak`
: Stops sending break characters.

`vdvrOpCodeStartLoopback`
: Starts loopback mode (not typically used for virtual drivers).

`vdvrOpCodeStopLoopback`
: Stops loopback mode (not typically used for virtual drivers).

`vdrvOpCodeFlushTxFIFO`
: Flushes the contents of the transmit FIFO.

`vdrvOpCodeFlushRxFIFO`
: Flushes the contents of the receive FIFO.

`vdrvOpCodeSendBufferedData`
: Notifies virtual device to send any buffered data it has not emptied from its internal buffers.

`vdrvOpCodeRcvCheckIdle`
: Called periodically to allow the virtual device time to check if there is data to be received. Because virtual devices execute in the same thread as applications, they can be prevented from handling notifications of received data.

`vdrvOpCodeEmuSetBlockingHook`
: Special op code for the Simulator.

`vdrvOpCodeGetOptTransmitSize`
: Returns the optimum buffer size for sending data or returns 0 to specify any buffer size is acceptable.

`vdrvOpCodeGetMaxRcvBlockSize`
: Returns the maximum receive block size that the Serial Manager should request from the virtual device. Can be used to implement flow control.

`vdrvOpCodeNotifyBytesReadFromQ`
: Tells the virtual device that some number of bytes have been read from the receive queue by the client application. Can be used to implement flow control.

`vdrvOpCodeSetDTRAsserted`
: Asserts or de-asserts the DTR signal.

`vdrvOpCodeGetDTRAsserted`
: Gets the status of the DTR signal.

`vdrvOpCodeWaitForConfiguration`
: Waits for USB enumeration to complete. Called from the send and receive functions of the Serial Manager. The driver should have a timeout for how long it waits for enumeration

to complete. The driver should return with no error if enumeration has already occurred or has occurred within the driver's timeout. If the enumeration has not occurred within the driver's timeout, the driver should return `serErrTimeOut`.

`vdrvOpCodeGetUSBDeviceDescriptor`
: Retrieves the device descriptor of a USB driver. Used to gather information about the device's capabilities. Implementation of this op code is optional. If the driver chooses to implement this op code, then the driver should return a pointer to the device descriptor. A driver that chooses not to implement this op code should return `serErrNotSupported`.

`vdrvOpCodeGetUSBConfigDescriptor`
: Retrieves the configuration descriptor of a USB driver. Used to gather information about the device's capabilities. Implementation of this op code is optional. If the driver chooses to implement this op code, then the driver should return a pointer to the device descriptor. A driver that chooses not to implement this op code should return `serErrNotSupported`.

`vdrvOpCodeEnableIRDA`
: Enable the IrDA mode and power up the IR line drivers.

`vdrvOpCodeDisableIRDA`
: Disable the IrDA mode and disable the IR line drivers.

`vdrvOpCodeEnableUART`
: Powers up the UART and the line drivers.

`vdrvOpCodeDisableUART`
: Powers down the UART and the line drivers.

`vdrvOpCodeRxEnable`
: Enables the receive FIFO, enables UART interrupts, and does whatever else is necessary to allow the UART to receive data.

`vdrvOpCodeRxDisable`
: Disables the receive FIFO and UART interrupts and does whatever is needed to prevent the UART from receiving data.

`vdrvOpCodeLineEnable`
: Enables the main serial line driver for the UART.

`vdrvOpCodeEnableUARTInterrupts`
: Enables the appropriate UART receive interrupts.

`vdrvOpCodeDisableUARTInterrupts`
: Disables all UART interrupts.

`vdrvOpCodeSetReceiveQueue`
: This op code is used by the Serial Manager to set the driver's receive queue. This control code is called when a driver that has previously been opened as a background port is opened as a fully open bidirectional port.

`vdrvOpCodeSaveState`
: Invoked when this port is yielded. This is a hook for the driver to save any current state.

`vdrvOpCodeRestoreState`
: Invoked when the foreground port is closed and this port can become the foreground port.

`vdrvOpCodeSetYieldPortCallback`
: Set the function to be called if the Serial Manager attempts to open another port when this one is open. This op code is for system use only.

`vdrvOpCodeSetYieldPortRefCon`
: Data to pass to the yield port callback function. System use only.

`vdvrOpCodeUserDef`
: User defined function invoked through `SrmControl()`.

`vdrvOpCodeSystem`
: Reserves op codes between 0x7000 and 0x8000 for system use.

`vdrvOpCodeCustom`
: Reserves op codes greater than 0x8000 for driver-specific use.

Compatibility

The op codes starting at `vdrvOpCodeWaitForConfiguration` are defined only if New Serial Manager Feature Set Version 2 is present. The op codes for yieldable ports and custom operations are defined only if both 4.0 New Feature Set is present as well.

Driver Constants

Driver Version Constants

The driver version constants specify which version of the driver API is implemented by this driver. The `DrvEntryPointProcPtr()` function passes this value back to the Serial Manager in the `drvrVersion` field of the `DrvrInfoType` function.

Constant	Value	Description
`kDrvrVersion`	4	The latest version of the API.
`kDrvrVersion3`	3	The version of the driver API that corresponds to New Serial Manager Feature Set Version 1 (which ships with roughly Palm OS® 3.3 up to Palm OS 4.0).
`kDrvrVersion4`	4	The version of the driver API that corresponds to New Serial Manager Feature Set Version 2 (which ships with roughly Palm OS 4.0 and higher).

Port Feature Constants

The port feature constants are flags that describe serial hardware capabilities.

Constant	Value	Description
`portPhysicalPort`	0x00000001	Should be set for a physical port, unset for a virtual port.
`portBkgndModeSupported`	0x00000002	Denotes that this port can be used for background ports. This flag is only applicable to virtual drivers. Background mode support is implied on physical drivers.

Constant	Value	Description
`portRS232Capable`	0x00000004	Set if this hardware has an RS-232 port.
`portIRDACapable`	0x00000008	Set if this hardware has an IR port and supports IrDA mode.
`portCradlePort`	0x00000010	Set if this hardware controls the cradle port.
`portExternalPort`	0x00000020	Set if this hardware port is external or on a memory card.
`portModemPort`	0x00000040	Set if this hardware communicates with a modem.
`portCncMgrVisible`	0x00000080	Set if this serial port's name is to be displayed in the Connection panel.
`portConsolePort`	0x00000100	Denotes this hardware controls the console port.
`portUSBCapable`	0x00000200	Set if this hardware has a USB port.
`portPrivateUse`	0x00001000	Set if this driver is for special software and not general applications.

Compatibility USB support is only available if New Serial Manager Feature Set Version 2 is present.

Virtual Driver-Defined Functions

The functions in this section must be defined by your virtual driver.

DrvEntryPointProcPtr Function

Purpose Entry point for the virtual driver.

Declared In `SerialDrvr.h`

Prototype

```
Err (*DrvEntryPointProcPtr)
    (DrvrEntryOpCodeEnum opCode, void *uartData)
```

Parameters

→ *opCode*
Entry function code.

↔ *uartData*
Pointer to data specific to `opCode`.

Returns

`errNone`
No error.

`-1`
The op code is invalid or the hardware could not be found.

Comments This function's purpose is based on the value of the `opCode` parameter. The three possible codes are `drvrEntryGetUartFeatures`, `drvrEntryGetDrvrFuncts`, and `drvrEntryGetUartFtrsNEntries`.

`DrvEntryPoint` is called with the `drvrEntryGetUartFeatures` code when the Serial Manager is installed into the system at boot time and is looking for all installed drivers. When this op code is set, the `uartData` pointer points to a `DrvrInfoType` structure. This function does not allocate the structure, it just fills in the fields with information.

This function should check to make sure the associated serial device can operate under the current OS and system settings. If the hardware cannot be found, the function should leave the `DrvrInfoType` struct untouched and return a -1 error.

The driver needs to supply a string that describes the port it manages. This string is displayed to the user in the Connection panel and is returned by the `SrmGetDeviceInfo()` function. To set this string, copy it into the `portDesc` field of the `DrvrInfoType` structure. Alternatively, you can supply this string in a driver resource of type 'tSTR' and id `kPortDescStrID`.

If the `DrvrInfoType` structure has a positive value in the `multipleEnumerations` field upon return, the Serial Manager

defines one port for each entry in the driver table. The `DrvEntryPoint` function is called again, this time with the `drvrEntryGetUartFtrsNEntries` code. The `uartData` pointer points to a new `DrvrInfoType` structure whose `multipleEnumerations` field indicates which port is to be defined. The function should supply all information specific to this port.

`DrvEntryPoint` is called with the `drvrEntryGetDrvrFuncts` code when a virtual port is opened. The `uartData` pointer points to a `VdrvAPIType` structure and `DrvEntryPoint` must fill in the fields of this structure with appropriate function pointers.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

The `drvrEntryGetUartFtrsNEntries` is only supported if New Serial Manager Feature Set Version 2 is present. This function is fully backwards compatible. Passing 0 for the `multipleEnumerations` field defines a single port for the driver.

VdrvCloseProcPtr Function

Purpose Handles all activities needed to close the virtual device.

Declared In `SerialDrvr.h`

Prototype

```
Err (*VdrvCloseProcPtr) (VdrvDataPtr drvrData)
```

Parameters

→ *`drvrData`*
Pointer to the driver's private global area.

Returns

`errNone`
No error.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

VdrvControlProcPtr Function

Purpose Extends the `SrmControl` function to the level of the virtual device.

Declared In `SerialDrvr.h`

Prototype

```
Err (*VdrvControlProcPtr) (VdrvDataPtr drvrData,
    VdrvCtlOpCodeEnum controlCode,
    void *controlData, UInt16 *controlDataLen)
```

Parameters

→ *drvrData*
Pointer to the driver's private global area.

→ *controlCode*
Control function op code. One of the op codes listed in the `VdrvCtlOpCodeEnum` type.

↔ *controlData*
Pointer to data for the specified control function.

↔ *controlDataLen*
Pointer to length of control data being passed in or out.

Returns

`errNone`
No error.

`serErrNotSupported`
`controlCode` not supported.

`serErrBadParam`
`controlData` or `controlDataLen` is bad.

Comments This function should support the op codes listed in the `VdrvCtlOpCodeEnum` type. If this function does not support an op code, it must return the `serErrNotSupported` error code for that op code.

Table 69.1 shows what is passed for the `controlData` and `controlDataLen` parameters for each of the control codes that use them. Control codes not listed do not use these parameters.

Table 69.1 VDrvControlProcPtr Parameters

`vdvrOpCodeSetBaudRate`	→ *controlData = Pointer to Int32 (baud rate),* → *controlDataLen = Pointer to sizeof(Int32).*
`vdvrOpCodeSetSettingsFlags`	→ *controlData = Pointer to UInt32 (bitfield; see Serial Settings Constants)* → *controlDataLen = Pointer to sizeof(UInt32)*
`vdvrOpCodeFIFOCount`	→ *controlData = Pointer to Int16, which contains the number of bytes in the FIFO.* → *controlDataLen = Pointer to sizeof(Int16).*
`vdrvOpCodeGetOptTransmitSize`	← *controlData = Pointer to Int32 (buffer size),* ← *controlDataLen = Pointer to sizeof(Int32).* Return the optimum buffer size for sending data, or 0 to specify any buffer size is acceptable.
`vdrvOpCodeGetMaxRcvBlockSize`	← *controlData = Pointer to Int32 (block size),* ← *controlDataLen = Pointer to sizeof(Int32).* Return the maximum block size that the Serial Manager should request from the virtual device.
`vdrvOpCodeNotifyBytes ReadFromQ`	→ *controlData = Pointer to Int32 (number of bytes read),* → *controlDataLen = Pointer to sizeof(Int32).*

Table 69.1 VDrvControlProcPtr Parameters *(continued)*

`vdrvOpCodeSetDTRAsserted`	→ *controlData = Pointer to Boolean indicating whether to enable or disable DTR.*
	→ *controlDataLen = Pointer to sizeof(Boolean)*
`vdrvOpCodeGetDTRAsserted`	← *controlData = Pointer to Boolean indicating whether DTR is enabled.*
	← *controlDataLen = Pointer to sizeof(Boolean)*
`vdvrOpCodeUserDef`	↔ *controlData = Pointer from SrmControl() (user-defined data),*
	↔ *controlDataLen = Pointer to sizeof(Int32).*

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

VdrvControlCustomProcPtr Function

Purpose Extends the `SrmCustomControl` function to the level of the virtual device.

Declared In `SerialDrvr.h`

Prototype

```
Err (*VdrvControlCustomProcPtr)
    (VdrvDataPtr drvrData, UInt16 opCode,
    UInt32 creator, void *controlData,
    void *controlDataLenP)
```

Parameters

→ *drvrData*
Pointer to the driver's private global area.

→ *controlCode*
Control function op code.

→ *creator*
Creator ID of the driver that defines the op code. The combination of creator ID and op code uniquely identifies the operation to be performed.

↔ *controlData*
Pointer to data for the specified control function.

↔ *controlDataLen*
Pointer to length of control data being passed in or out.

Returns

`errNone`
No error.

`serErrNotSupported`
`controlCode` not supported.

`serErrBadParam`
`controlData` or `controlDataLen` is bad.

Comments

This function is a mechanism for a virtual driver to create control codes specific to that driver, allowing for the support of new technologies that have interfaces through the Serial Manager.

Compatibility

Implemented only if both New Serial Manager Feature Set Version 2 and 4.0 New Feature Set are present.

VdrvOpenProcPtr Function

Purpose

Initializes the virtual device to begin communication.

Declared In

`SerialDrvr.h`

Prototype

```
Err (*VdrvOpenProcPtr) (VdrvDataPtr *drvrData,
   UInt32 baudRate, DrvrHWRcvQPtr rcvQP)
```

Parameters

↔ *drvrData*
Pointer to a pointer to the driver's private global area (allocated by this function). A pointer to this private global area is passed to the other virtual driver functions.

→ *baudRate*
Initial baud rate setting.

→ *rcvQP*
Pointer to the driver's receive queue buffer structure. For details on the fields, see `DrvrRcvQType`.

Returns `errNone`
No error.

Comments This function must allocate and initialize any global variables (and pass back a pointer to a pointer to them in `drvrDataP`), do any set-up necessary for communicating with other software, and save the `rcvQP` pointer since it will need the functions and pointers to structures enclosed within to be able to save received data into the Serial Manager's receive queue.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

VdrvOpenProcV4Ptr Function

Purpose Initializes the virtual device to begin communication.

Declared In `SerialDrvr.h`

Prototype

```
Err (*VdrvOpenProcV4Ptr) (VdrvDataPtr *drvrData,
   VdrvConfigPtr configP, DrvrHWRcvQPtr rcvQP)
```

Parameters

↔ *drvrData*
Pointer to a pointer to the driver's private global area (allocated by this function). A pointer to this private global area is passed to the other virtual driver functions.

→ *configP*
Pointer to the configuration structure specifying the port's properties. See `VdrvConfigType`.

→ *rcvQP*
Pointer to the driver's receive queue buffer structure. For details on the fields, see `DrvrRcvQType`.

Returns `errNone`
No error.

Comments This function must allocate and initialize any global variables (and pass back a pointer to a pointer to them in `drvrDataP`), do any set-up necessary for communicating with other software, and save the `rcvQP` pointer since it will need the functions and pointers to structures enclosed within to be able to save received data into the Serial Manager's receive queue.

Compatibility Implemented only if both New Serial Manager Feature Set Version 2 and 4.0 New Feature Set are present.

VdrvStatusProcPtr Function

Purpose Returns virtual device status.

Declared In `SerialDrvr.h`

Prototype

```
UInt16 (*VDrvStatusProcPtr) (VdrvDataPtr drvrData)
```

Parameters

→ *drvrData*
Pointer to the driver's private global area.

Returns An unsigned long bitfield denoting the status of the virtual device, but only if the virtual device is emulating hardware. The individual bit flags are described in the `DrvrStatusEnum` type.

Comments Generally, status is returned only to the client application using the virtual device. The Serial Manager does not use status information from virtual devices.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

VdrvWriteProcPtr Function

Purpose Writes a block of bytes.

Declared In `SerialDrvr.h`

Prototype

```
UInt32 (*VdrvWriteProcPtr) (VdrvDataPtr drvrDataP,
    void *bufP, UInt32 size, Err *errP)
```

Parameters

→ *drvrDataP*
Pointer to the driver's private global area.

→ *bufP*
Pointer to buffer containing the data to be written to the virtual device.

→ *size*
Number of bytes in the buffer `bufP`.

← *errP*
Pointer to an error code resulting from the operation. Zero is returned if there is no error.

Returns Returns the actual number of bytes written.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

Serial Manager Queue Functions

The functions in this section are supplied by the Serial Manager to the virtual driver through the `DrvrRcvQType` passed to the `VdrvOpenProcPtr()` function.

GetSizeProcPtr Function

Purpose Returns the total size of the Serial Manager's receive queue.

Declared In `SerialDrvr.h`

Prototype

```
typedef UInt32 (*GetSizeProcPtr) (void *theQ)
```

Parameters → *theQ*
Pointer to the receive queue.

Returns Size in bytes of the Serial Manager's receive queue.

Comments This function is useful for implementing flow control.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

GetSpaceProcPtr Function

Purpose Returns the available space in the Serial Manager's receive queue.

Declared In `SerialDrvr.h`

Prototype

```
typedef UInt32 (*GetSpaceProcPtr) (void *theQ)
```

Parameters → *theQ*
Pointer to the receive queue.

Returns Size in bytes of the available space in the Serial Manager's receive queue.

Comments This function is useful for implementing flow control.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

SignalCheckPtr Function

Purpose Check the queue to see if the semaphore needs to be signalled.

Declared In `SerialDrvr.h`

Prototype

```
typedef void (*SignalCheckPtr) (void *theQ,
    UInt16 lineErrsP)
```

Parameters

→ *theQ*
Pointer to the receive queue.

→ *lineErrsP*
Any serial line errors received should be reported here.

Returns Returns nothing.

Comments This function signals that there is data to be received without writing anything to the receive queue. The `WriteByteProcPtr()` and `WriteBlockProcPtr()` functions also signal that there is data to be received after they have written the data to the queue.

Compatibility Implemented only if New Serial Manager Feature Set Version 2 is present.

WriteBlockProcPtr Function

Purpose Writes a block of bytes to the Serial Manager's receive queue.

Declared In `SerialDrvr.h`

Prototype

```
typedef Err (*WriteBlockProcPtr) (void *theQ,
    UInt8 *bufP, UInt16 size, UInt16 lineErrs)
```

Parameters

→ *theQ*
Pointer to the receive queue.

→ *bufP*
Pointer to the buffer holding bytes to be written.

→ *size*
Size of `bufP`.

→ *lineErrs*
Any serial line errors received should be reported here.

Returns

`errNone`
No error.

`serErrLineErr`
There was a software overrun line error.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

WriteByteProcPtr Function

Purpose Writes one byte to the Serial Manager's receive queue.

Declared In `SerialDrvr.h`

Prototype

```
typedef Err (*WriteByteProcPtr) (void *theQ,
   UInt8 theByte, UInt16 lineErrs)
```

Parameters

→ *theQ*
Pointer to the receive queue.

→ *theByte*
The byte to be written to the queue.

→ *lineErrs*
Any serial line errors received should be reported here.

Returns

`errNone`
No error.

`serErrLineErr`
There was a software overrun line error.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

70

Serial Manager

This chapter provides reference material for the Serial Manager API:

- Serial Manager Data Structures
- Serial Manager Constants
- Serial Manager Functions
- Serial Manager Application-Defined Functions

The header file `SerialMgr.h` declares the Serial Manager API. The file `SystemResources.h` defines some serial port constants. For more information on the Serial Manager, see the chapter "Serial Communication" on page 89 of the *Palm OS Programmer's Companion*, vol. II, *Communications*.

Serial Manager Data Structures

DeviceInfoType Struct

Purpose The `DeviceInfoType` structure defines information about a serial device. This structure is returned by the `SrmGetDeviceInfo()` function.

Prototype

```
typedef struct DeviceInfoType {
  UInt32 serDevCreator;
  UInt32 serDevFtrInfo;
  UInt32 serDevMaxBaudRate;
  UInt32 serDevHandshakeBaud;
  Char *serDevPortInfoStr;
  UInt8 reserved[8];
} DeviceInfoType;
typedef DeviceInfoType *DeviceInfoPtr;
```

Fields

`serDevCreator`
: Four-character creator ID for serial driver.

`serDevFtrInfo`
: Flags defining features of this serial hardware. See Serial Capabilities Constants for a description of these flags.

`serDevMaxBaudRate`
: Maximum baud rate for this device.

`serDevHandshakeBaud`
: Hardware handshaking is recommended for baud rates over this rate.

`serDevPortInfoStr`
: Description of serial hardware device or virtual device.

SrmCtlEnum Enum

Purpose

The `SrmCtlEnum` enumerated type specifies a serial control operation. Specify one of these enumerated types for the `op` parameter to the `SrmControl()` call.

Prototype

```
typedef enum SrmCtlEnum {
  srmCtlFirstReserved = 0,
  srmCtlSetBaudRate,
  srmCtlGetBaudRate,
  srmCtlSetFlags,
  srmCtlGetFlags,
  srmCtlSetCtsTimeout,
  srmCtlGetCtsTimeout,
  srmCtlStartBreak,
  srmCtlStopBreak,
  srmCtlStartLocalLoopback,
  srmCtlStopLocalLoopback,
  srmCtlIrDAEnable,
  srmCtlIrDADisable,
  srmCtlRxEnable,
  srmCtlRxDisable,
  srmCtlEmuSetBlockingHook,
  srmCtlUserDef,
  srmCtlGetOptimalTransmitSize,
  srmCtlSetDTRAsserted,
  srmCtlGetDTRAsserted,
  srmCtlSetYieldPortCallback,
```

```
  srmCtlSetYieldPortRefCon,
  srmCtlSystemReserved = 0x7000
  srmCtlCustom = 0x8000,
  srmCtlLAST
} SrmCtlEnum;
```

Constants

`srmCtlSetBaudRate`
: Sets the current baud rate for the serial hardware.

`srmCtlGetBaudRate`
: Gets the current baud rate for the serial hardware.

`srmCtlSetFlags`
: Sets the current flag settings for the serial hardware. Specify flags from the set described in Serial Settings Constants.

`srmCtlGetFlags`
: Gets the current flag settings for the serial hardware.

`srmCtlSetCtsTimeout`
: Sets the current CTS timeout value for hardware handshaking.

`srmCtlGetCtsTimeout`
: Gets the current CTS timeout value for hardware handshaking.

`srmCtlStartBreak`
: Turn RS-232 break signal on. Caller is responsible for turning this signal on and off and insuring it is on long enough to generate a viable break.

`srmCtlStopBreak`
: Turn RS-232 break signal off.

`srmCtlStartLocalLoopback`
: Start local loopback test.

`srmCtlStopLocalLoopback`
: Stop local loopback test.

`srmCtlIrDAEnable`
: Enable IrDA connection on this serial port.

NOTE: You cannot enable an IrDA connection on a VZ processor.

`srmCtlIrDADisable`
: Disable IrDA connection on this serial port.

`srmCtlRxEnable`
: Enable receiver (for IrDA).

`srmCtlRxDisable`
: Disable receiver (for IrDA).

`srmCtlEmuSetBlockingHook`
: Set a blocking hook routine for emulation mode only. Not supported on the actual device.

`srmCtlUserDef`
: This is a user-defined function that third-party hardware developers can use to set or retrieve hardware-specific information from the serial driver. This op code invokes the driver's corresponding control function with its user-defined op code and the parameters are passed directly through to the serial driver. A serial driver that does not handle this function returns a `serErrBadParam` error.

  The `srmCtlUserDef` op code is superseded by defining a custom op code if New Serial Manager Feature Set Version 2 is present.

`srmCtlGetOptimalTransmitSize`
: Ask the port for the most efficient buffer size for transmitting data packets. This op code returns an error (buffering not necessary), 0 (buffering requested, but application can choose buffer size), or a number greater than 0 (recommended buffer size).

`srmCtlSetDTRAsserted`
: Enabled or disable the DTR signal. This is not supported by all hardware.

`srmCtlGetDTRAsserted`
: Ask the port whether the DTR signal is enabled or disabled.

`srmCtlSetYieldPortCallback`
: Set the function to be called if the Serial Manager attempts to open another port when this one is open. This op code is for system use only.

`srmCtlSetYieldPortRefCon`
: Data to pass to the yield port callback function. System use only.

`srmCtlSystemReserved`
: Reserves op codes between 0x7000 and 0x8000 for system use.

`srmCtlCustom`
: Reserves op codes greater than 0x8000 for driver-specific use.

Compatibility Custom control op codes are only supported if both New Serial Manager Feature Set Version 2 and 4.0 New Feature Set are present.

SrmOpenConfigType Struct

Purpose The `SrmOpenConfigType` structure specifies parameters for opening a serial port. This structure is passed as a parameter to `SrmExtOpen()`.

Prototype

```
typedef struct SrmOpenConfigType {
  UInt32 baud;
  UInt32 function;
  MemPtr drvrDataP;
  UInt16 drvrDataSize;
  UInt32 sysReserved1;
  UInt32 sysReserved2;
} SrmOpenConfigType;
```

Fields

`baud`
: Baud rate at which to open the connection. Serial drivers that do not require baud rates ignore this field.

`function`
: The reason why the port was opened. Specify the creator ID of the application that is opening the port or one of the following values:

  `serFncUndefined`
  : Undefined function. This is the default value for this field.

  `serFncPPPSession`
  : The connection is to be used for the PPP interface.

  `serFncSLIPSession`
  : The connection is to be used for the SLIP session.

  `serFncDebugger`
  : The connection is to be used for a debugging session.

`serFncHotSync`
The connection is to be used for a HotSync operation.

`serFncConsole`
The connection is to the debugging console.

`serFncTelephony`
The connection is to the telephony library.

The `function` field is used by protocols such as USB and Bluetooth that perform different setup tasks based on which type of application is using them. RS-232 drivers ignore this parameter.

`drvrDataP`
Pointer to a driver-specific data block.

`drvrDataSize`
The size of the data block pointed to by `drvrDataP`.

`sysReserved1`
Reserved for future use.

`sysReserved2`
Reserved for future use.

Compatibility This structure is only defined if both New Serial Manager Feature Set Version 2 and 4.0 New Feature Set are present.

Serial Manager Constants

Port Constants

When you specify the port to open in the `SrmOpen()`, `SrmOpenBackground()`, `SrmExtOpen()`, or `SrmExtOpenBackground()` call, you can use either a logical port constant, physical port constant, or a virtual port constant, but it is highly recommended that you use a logical port constant wherever possible.

Logical Serial Port Constants

These constants specify the logical port names.

Constant	Value	Description
`serPortLocalHotSync`	0x8000	The physical HotSync port. The Serial Manager automatically detects whether this port is USB or RS-232.
`serPortCradlePort`	0x8000	Cradle port. The Serial Manager automatically detects whether this port is USB or RS-232. Most applications should specify this as the port.
`serPortIrPort`	0x8001	The IR port. This is a raw IrDA port with no protocol support.
`serPortConsolePort`	0x8002	The debug console port, either USB or RS-232. USB is preferred where both are available.
`serPortCradleRS232Port`	0x8003	Port for the RS-232 cradle. Specify this port if you want to ensure that your application uses RS-232 communications only.
`serPortCradleUSBPort`	0x8004	Port for the USB cradle. Specify this port if you want to ensure that your application uses USB communications only.

Compatibility USB ports are only supported if New Serial Manager Feature Set Version 2 is present.

Physical Serial Port Constants

The physical port constants specify 4-character constants that reference the physical hardware of the device. Doing so is *not* recommended because the hardware they reference may not exist on a particular device.

Physical port	Value	Description
`sysFileCUart328`	'u328'	Cradle port using the 68328 UART. This port can be switched between RS232 and IrDA mode using the `SrmControl()` call.
`sysFileCUart328EZ`	'u8EZ'	Cradle port using the 68328EZ UART. This port can also be switched between RS232 and IrDA mode.
`sysFileCUart650`	'u650'	Specifies the IR port on the upgrade card for Palm Personal or Palm Professional devices. This gives you a raw IR port like calling `SrmControl` does, but it only exists on devices that have the upgrade card.

Virtual Serial Port Constants

The virtual port constants specify 4-character constants that identify virtual ports, simulating a hardware interface. Virtual ports are not tied to specific hardware.

Physical port	Value	Description
`sysFileCVirtIrComm`	'ircm'	A virtual serial cable over an IrDA link using the IRComm protocol. It can only be used to talk to another IRComm device.
`sysFileCVirtRfComm`	'rfcm'	RFCOMM (Bluetooth) virtual port plug-in.
`sysFileCBtConnectPanelHelper`	'btcp'	Bluetooth Connection Panel helper application.

Compatibility All virtual port constants other than `sysFileCVirtIrComm` are only defined if both New Serial Manager Feature Set Version 2 and 4.0 New Feature Set are present, and if the 5.0 New Feature Set is *not* present.

Serial Capabilities Constants

The serial capabilities constant flags describe serial hardware capabilities. These flags are set in the `serDevFtrInfo` field of the `DeviceInfoType` structure.

Constant	Value	Description
`serDevCradlePort`	0x00000001	Serial hardware controls RS-232 serial from cradle connector of Palm device.
`serDevRS232Serial`	0x00000002	Serial hardware has RS-232 line drivers.
`serDevIRDACapable`	0x00000004	Serial hardware has IR line drivers and generates IrDA mode serial signals.
`serDevModemPort`	0x00000008	Serial hardware drives modem connection.
`serDevCncMgrVisible`	0x00000010	Serial device port name string is to be displayed in the Connection panel.
`serDevConsolePort`	0x00000020	Serial device is the default console port.
`serDevUSBCapable`	0x00000040	Serial hardware controls USB serial from cradle connector of Palm device.

Compatibility USB ports are only supported if New Serial Manager Feature Set Version 2 is present.

Serial Settings Constants

The serial settings constants identify bit flags that correspond to various serial hardware settings. Use `SrmControl()` with the op code `srmCtlSetFlags` to control which settings are used.

Constant	Value	Description
`srmSettingsFlagStopBitsM`	0x00000001	Mask for stop bits field
`srmSettingsFlagStopBits1`	0x00000000	1 stop bit (default)
`srmSettingsFlagStopBits2`	0x00000001	2 stop bits
`srmSettingsFlagParityOnM`	0x00000002	Mask for parity on
`srmSettingsFlagParityEvenM`	0x00000004	Mask for parity even
`srmSettingsFlagXonXoffM`	0x00000008	Mask for Xon/Xoff flow control (not implemented)
`srmSettingsFlagRTSAutoM`	0x00000010	Mask for RTS receive flow control. This is the default.
`srmSettingsFlagCTSAutoM`	0x00000020	Mask for CTS transmit flow control
`srmSettingsFlagBitsPerCharM`	0x000000C0	Mask for bits per character
`srmSettingsFlagBitsPerChar5`	0x00000000	5 bits per character
`srmSettingsFlagBitsPerChar6`	0x00000040	6 bits per character
`srmSettingsFlagBitsPerChar7`	0x00000080	7 bits per character
`srmSettingsFlagBitsPerChar8`	0x000000C0	8 bits per character (default)

`srmSettingsFlagBitsPerChar7`	0x00000080	7 bits per character
`srmSettingsFlagBitsPerChar8`	0x000000C0	8 bits per character (default)

Status Constants

The status constants identify bit flags that correspond to the status of serial signals. They can be returned by the `SrmGetStatus()` function.

Constant	Value	Description
`srmStatusCtsOn`	0x00000001	CTS line is active.
`srmStatusRtsOn`	0x00000002	RTS line is active.
`srmStatusDsrOn`	0x00000004	DSR line is active.
`srmStatusBreakSigOn`	0x00000008	Break signal is active.

Line Error Constants

The line error constants identify bit flags that correspond to the line errors that may occur on the port. They can be returned by the `SrmGetStatus()` function.

Constant	Value	Description
`serLineErrorParity`	0x0001	Parity error
`serLineErrorHWOverrun`	0x0002	Hardware overrun
`serLineErrorFraming`	0x0004	Framing error
`serLineErrorBreak`	0x0008	Break signal asserted
`serLineErrorHShake`	0x0010	Line handshake error
`serLineErrorSWOverrun`	0x0020	Software overrun
`serLineErrorCarrierLost`	0x0040	Carrier detect signal dropped

Serial Manager Functions

SrmClearErr Function

Purpose Clears the port of any line errors.

Declared In `SerialMgr.h`

Prototype

```
Err SrmClearErr (UInt16 portId)
```

Parameters

→ *portID*
Port ID returned from SrmOpen() or SrmExtOpen().

Returns This function returns the following error codes:

`errNone`
No error.

`serErrNotSupported`
The port is not the foreground port.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

SrmClose Function

Purpose Closes a serial port and makes it available to other applications, regardless of whether the port is a foreground or background port.

Declared In `SerialMgr.h`

Prototype

```
Err SrmClose (UInt16 portId)
```

Parameters

→ *portId*
Port ID for port to be closed.

Returns This function returns the following error codes:

`errNone`
No error.

`serErrBadPort`
This port doesn't exist.

`serErrNotOpen`
The serial port is not open.

`serErrNoDevicesAvail`
No serial devices could be found.

Comments If a foreground port is being closed and a background port exists, the background will have access to the port as long as another foreground port is not opened.

If a foreground port is being closed and a yielded port exists, the yielded port will have access to the port as long as it does not yield to the opening of another foreground port. If there are both a yielded port and a background port for the foreground port being closed, the yielded port takes precedence over the background port.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmOpen(), SrmOpenBackground()

SrmControl Function

Purpose Performs a serial control function.

Declared In `SerialMgr.h`

Prototype

```
Err SrmControl (UInt16 portId, UInt16 op,
   void *valueP, UInt16 *valueLenP)
```

Parameters

→ *portID*
Port ID returned from SrmOpen() or SrmExtOpen().

→ *op*
Control operation to perform. Specify one of the `SrmCtlEnum` enumerated types.

↔ *valueP*
Pointer to a value to use for the operation. See Comments for details.

↔ *valueLenP*
Pointer to the size of `*valueP`. See Comments for details.

Returns This function returns the following error codes:

`errNone`
No error.

`serErrBadParam`
An invalid op code was specified.

`serErrBadPort`
This port doesn't exist.

`serErrNotOpen`
The serial port is not open.

`serErrNoDevicesAvail`
No serial devices could be found.

`serErrNotSupported`
The specified op code is not supported in the current configuration.

Comments

Table 70.1 shows what to pass for the `valueP` and `valueLenP` parameters for each of the operation codes. Control codes not listed do not use these parameters. See `SrmCtlEnum` for a complete list of control codes.

Table 70.1 SrmControl Parameters

Operation Code	Parameters
`srmCtlSetBaudRate`	→ *valueP = Pointer to Int32 (baud rate)*
	→ *valueLenP = Pointer to sizeof(Int32)*
`srmCtlGetBaudRate`	← *valueP = Pointer to Int32 (baud rate)*
	← *valueLenP = Pointer to Int16*
`srmCtlSetFlags`	→ *valueP = Pointer to Uint32 (bitfield; see Serial Settings Constants)*
	→ *valueLenP = Pointer to sizeof(UInt32)*
`srmCtlGetFlags`	← *valueP = Pointer to UInt32 (bitfield)*
	← *valueLenP = Pointer to Int16*

Table 70.1 SrmControl Parameters *(continued)*

Operation Code	Parameters
`srmCtlSetCtsTimeout`	→ *`valueP = Pointer to Int32 (timeout value)`* → *`valueLenP = Pointer to sizeof(Int32)`*
`srmCtlGetCtsTimeout`	← *`valueP = Pointer to Int32 (timeout value)`* ← *`valueLenP = Pointer to Int16`*
`srmCtlUserDef`	↔ *`valueP = Pointer passed to the serial or virtual driver`* ↔ *`valueLenP = Pointer to sizeof(Int32)`* For a serial driver, these pointers are passed to the driver's control function and they contain that functions return values (if any) upon return.
`srmCtlGetOptimalTransmitSize`	← *`valueP = Pointer to Int32`* ← *`valueLenP = Pointer to sizeof(Int32)`* If an error is returned by `SrmControl`, no buffering should be done. If `valueP` points to zero, buffering is requested, but the transmitting application cannot determine the buffer size. If `valueP` points to a number > 0, then try to send data in blocks of this number of bytes, as this is the most efficient block size for this particular device.

Table 70.1 SrmControl Parameters *(continued)*

Operation Code	Parameters
`srmCtlSetDTRAsserted`	→ *valueP = Pointer to Boolean indicating whether to enable or disable DTR.*
	→ *valueLenP = Pointer to sizeof(Boolean)*
`srmCtlGetDTRAsserted`	← *valueP = Pointer to Boolean indicating whether DTR is enabled.*
	← *valueLenP = Pointer to Int16*

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also `SrmCustomControl()`

SrmCustomControl Function

Purpose Performs a custom serial control function.

Declared In `SerialMgr.h`

Prototype

```
Err SrmCustomControl (UInt16 portId,
   UInt16 opCode, UInt32 creator, void *valueP,
   UInt16 *valueLenP)
```

Parameters

→ *portID*
Port ID returned from SrmOpen() or SrmExtOpen().

→ *opCode*
Control operation to perform. The op code must be greater than `srmCtlCustom`.

→ *creator*
Creator ID of the driver that defines the op code. The combination of creator ID and op code uniquely identifies the operation to be performed.

↔ *valueP*
Pointer to a value to use for the operation.

↔ *valueLenP*
Pointer to the size of `*valueP`.

Returns This function returns the following error codes:

`errNone`
No error.

`serErrNotSupported`
The port is not the foreground port.

`serErrBadPort`
This port doesn't exist.

`serErrNotOpen`
The serial port is not open.

`serErrNoDevicesAvail`
No serial devices could be found.

Comments This function is a mechanism for a virtual driver to create control codes specific to that driver, allowing for the support of new technologies that have interfaces through the Serial Manager.

This function simply forwards the `opCode` and any `valueP` parameter to the virtual driver for the port. The virtual driver may return its own error code if the `opCode` or the input in `valueP` is invalid.

Compatibility Implemented only if both New Serial Manager Feature Set Version 2 and 4.0 New Feature Set are present.

See Also `SrmControl()`

SrmExtOpen Function

Purpose Opens a foreground port connection with the specified configuration.

Declared In `SerialMgr.h`

Prototype

```
Err SrmExtOpen (UInt32 port,
   SrmOpenConfigType *configP, UInt16 configSize,
   UInt16 *newPortIdP)
```

Parameters → *port*
The four-character port name (such as 'ircm' or 'u328') or logical port number to be opened. (See Port Constants.)

→ *configP*
Pointer to the configuration structure specifying the serial port's properties. See `SrmOpenConfigType`.

→ *configSize*
The size of the configuration structure pointed to by `configP`.

← *newPortIdP*
Contains the port ID to be passed to other Serial Manager functions.

Returns This function returns the following error codes:

`errNone`
No error.

`serErrBadPort`
The `port` parameter does not specify a valid port.

`serErrBadParam`
The `configP` parameter is `NULL`.

`serErrAlreadyOpen`
The Serial Manager already has a port open.

`memErrNotEnoughSpace`
There was not enough memory available to open the port.

Comments Do not keep the port open any longer than necessary. An open serial port consumes more energy from the device's batteries.

The values specified in the `configP` parameter depend on the type of connection being made. For RS-232 connections, you specify the baud rate but not a purpose. For USB connections, you specify a purpose but not a baud rate.

A newly opened port has its line errors cleared, the default CTS timeout set (specified by the constant `srmDefaultCTSTimeout`), a 512-byte receive queue allocated, 1 stop bit, 8 bits per character, RTS enabled, and flow control enabled. To increase the receive queue size, use `SrmSetReceiveBuffer()`. To change the other serial port settings, use `SrmControl()`.

Compatibility Implemented only if both New Serial Manager Feature Set Version 2 and 4.0 New Feature Set are present. The `SrmExtOpen` function replaces the `SrmOpen()` function.

See Also `SrmOpen()`, `SrmExtOpenBackground()`

SrmExtOpenBackground Function

Purpose Opens a port with the specified configuration in the background. Background ports relinquish control when another task opens the port with the `SrmOpen()` or `SrmExtOpen()` call.

Declared In `SerialMgr.h`

Prototype

```
Err SrmExtOpenBackground (UInt32 port,
    SrmOpenConfigType *configP, UInt16 configSize,
    UInt16 *newPortIdP)
```

Parameters

→ *port*
: Physical or logical port number to be opened. See Port Constants for more information.

→ *configP*
: Pointer to the configuration structure specifying the serial port's properties. See `SrmOpenConfigType`.

→ *configSize*
: The size of the configuration structure pointed to by `configP`.

← *newPortIdP*
: Contains the port ID to be passed to other Serial Manager functions.

Returns This function returns the following error codes:

`errNone`
: No error.

`serErrAlreadyOpen`
: This port already has an installed background owner.

`serErrBadPort`
: This port doesn't exist.

`serErrNotSupported`
: This type of port cannot be opened in the background.

`serErrBadParam`
The `configP` parameter is `NULL`.

`memErrNotEnoughSpace`
There was not enough memory available to open the port.

Comments This function is provided to support tasks that want to use a serial device to receive data only when no other task is using the port.

If a background port is forced to surrender control of the hardware as a result of another task opening a foreground connection, all buffers for the background port are flushed. After this active task closes the port, active control of the port is returned to the background task. Only one task can have background ownership of the port.

Note that background ports have limited functionality: they can only receive data and notify owning clients of what data has been received.

The values specified in the `configP` parameter depend on the type of connection being made. For RS-232 connections, you specify the baud rate but not a purpose. For USB connections, you specify a purpose but not a baud rate.

Compatibility Implemented only if both New Serial Manager Feature Set Version 2 and 4.0 New Feature Set are present. The `SrmExtOpenBackground` function replaces the `SrmOpenBackground()` function.

See Also SrmOpen(), SrmExtOpen()

SrmGetDeviceCount Function

Purpose Returns the number of available serial devices.

Declared In `SerialMgr.h`

Prototype
```
Err SrmGetDeviceCount (UInt16 *numOfDevicesP)
```

Parameters ← *`numOfDevicesP`*
Pointer to address where the number of serial devices is returned.

Returns `errNone`
No error.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmGetDeviceInfo()

SrmGetDeviceInfo Function

Purpose Returns information about a serial device.

Declared In `SerialMgr.h`

Prototype

```
Err SrmGetDeviceInfo (UInt32 deviceID,
   DeviceInfoType *deviceInfoP)
```

Parameters

→ *deviceID*
: ID of serial device to get information for. You can pass a zero-based index (0, 1, 2, ...), a valid port ID returned from `SrmOpen()` or `SrmExtOpen()`, or a 4-character port name (such as 'u328', 'u650', or 'ircm'). See Port Constants.

← *deviceInfoP*
: Pointer to a `DeviceInfoType` structure where information about the device is returned.

Returns This function returns the following error codes:

`errNone`
: No error.

`serErrBadPort`
: This port doesn't exist.

`serErrNoDevicesAvail`
: The Serial Manager cannot find any serial devices.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmGetDeviceCount()

SrmGetStatus Function

Purpose Returns status information about the serial hardware.

Declared In `SerialMgr.h`

Prototype

```
Err SrmGetStatus (UInt16 portId,
   UInt32 *statusFieldP, UInt16 *lineErrsP)
```

Parameters

→ *portID*
: Port ID returned from SrmOpen() or SrmExtOpen().

← *statusFieldP*
: Pointer to address where hardware status information for the port is returned. This is a 32-bit field using the flags described in Status Constants.

← *lineErrsP*
: Pointer to address where the number of line errors for the port is returned. The line error flags are described in Line Error Constants.

Returns This function returns the following error codes:

`errNone`
: No error.

`serErrBadPort`
: This port doesn't exist.

`serErrNotSupported`
: The port is a yielded port.

`serErrNoDevicesAvail`
: No serial devices could be found.

Comments Typically, `SrmGetStatus` is called to retrieve the line errors for the port if some of the send and receive functions return a `serErrLineErr` error code.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

SrmOpen Function

Purpose Opens a foreground port connection with the specified port name or logical port number.

Declared In `SerialMgr.h`

Prototype

```
Err SrmOpen (UInt32 port, UInt32 baud,
    UInt16 *newPortIdP)
```

Parameters

→ *port*
: The four-character port name or logical port number to be opened. See Port Constants for more information.

→ *baud*
: Initial baud rate of port.

← *newPortIdP*
: Contains the port ID to be passed to other Serial Manager functions.

Returns This function returns the following error codes:

`errNone`
: No error.

`serErrAlreadyOpen`
: This port already has an installed foreground owner.

`serErrBadPort`
: This port doesn't exist.

`memErrNotEnoughSpace`
: There was not enough memory available to open the port.

Comments Only one application or task may have access to a particular serial port at any time.

Do not keep the port open any longer than necessary. An open serial port consumes more energy from the device's batteries.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

If New Serial Manager Feature Set Version 2 is present, the `SrmOpen` function is replaced by `SrmExtOpen()`. `SrmOpen` is supported for backward compatibility.

See Also SrmOpenBackground()

SrmOpenBackground Function

Purpose Allows a task to open, initialize, and use the port, but always relinquishes control of the port when another task opens the port with the `SrmOpen()` call.

Declared In `SerialMgr.h`

Prototype

```
Err SrmOpenBackground (UInt32 port, UInt32 baud,
    UInt16 *newPortIdP)
```

Parameters

→ *port*
The four-character port name or logical port number to be opened. See Port Constants for more information.

→ *baud*
Initial baud rate of port.

← *newPortIdP*
Contains the port ID to be passed to other Serial Manager functions.

Returns This function returns the following error codes:

`errNone`
No error.

`serErrAlreadyOpen`
This port already has an installed background owner.

`serErrBadPort`
This port doesn't exist.

`memErrNotEnoughSpace`
There was not enough memory available to open the port.

Comments This function is provided to support tasks that want to use a serial device to receive data only when no other task is using the port.

If a background port is forced to surrender control of the hardware as a result of another task opening a foreground connection, all buffers for the background port are flushed. After this active task closes the port, active control of the port is returned to the background task. Only one task can have background ownership of the port.

Note that background ports have limited functionality: they can only receive data and notify owning clients of what data has been received.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

If New Serial Manager Feature Set Version 2 is present, the `SrmOpenBackground()` function is replaced by `SrmExtOpenBackground()`. `SrmOpenBackground()` is supported for backward compatibility. If Palm OS Cobalt Feature Set is present, this function is unimplemented.

See Also SrmOpen()

SrmPrimeWakeupHandler Function

Purpose Sets the number of received bytes that triggers a call to the wakeup handler function.

Declared In `SerialMgr.h`

Prototype

```
Err SrmPrimeWakeupHandler (UInt16 portId,
   UInt16 minBytes)
```

Parameters

→ *portId*
Port ID returned from SrmOpen() or SrmExtOpen().

→ *minBytes*
Number of bytes that must be received before wakeup handler is called. Typically, this is set to 1.

Returns This function returns the following error codes:

`errNone`
No error.

`serErrBadPort`
This port doesn't exist.

`serErrNotOpen`
The port is not open.

`serErrNoDevicesAvail`
No serial devices could be found.

Comments This function primes a wakeup handler installed by `SrmSetWakeupHandler()`.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmSetWakeupHandler(), WakeupHandlerProcPtr()

SrmReceive Function

Purpose Receives a specified number of bytes.

Declared In `SerialMgr.h`

Prototype

```
UInt32 SrmReceive (UInt16 portId, void *rcvBufP,
   UInt32 count, Int32 timeout, Err *errP)
```

Parameters

→ *portID*
Port ID returned from SrmOpen() or SrmExtOpen().

← *rcvBufP*
Pointer to buffer where received data is to be returned.

→ *count*
Length of data buffer (in bytes). This specifies the number of bytes to receive.

→ *timeout*
The amount of time (in ticks) that the Serial Manager waits to receive the requested block of data. At the end of the timeout, data received up to that time is returned.

← *errP*
Error code.

Returns Number of bytes of data actually received.

Comments The following error codes can be returned in `errP`:

`errNone`
No error.

`serErrBadPort`
This port doesn't exist.

`serErrNotOpen`
The port is not open.

`serErrTimeOut`
Unable to receive data within the specified timeout period.

`serErrConfigurationFailed`
The port needs time to configure, and the configuration has failed.

`serErrNotSupported`
The port is not the foreground port.

`serErrConfigurationFailed`
The port could not configure itself.

`serErrLineErr`
A line error occurred during the receipt of data. Use `SrmGetStatus()` to obtain the exact line error.

`serErrNoDevicesAvail`
No serial devices could be found.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmReceiveCheck(), SrmReceiveFlush(), SrmReceiveWait()

SrmReceiveCheck Function

Purpose Checks the receive FIFO and returns the number of bytes in the serial receive queue.

Declared In `SerialMgr.h`

Prototype

```
Err SrmReceiveCheck (UInt16 portId,
   UInt32 *numBytesP)
```

Parameters

→ *portID*
Port ID returned from SrmOpen() or SrmExtOpen().

← *numBytesP*
Number of bytes in the receive queue.

Returns This function returns the following error codes:

`errNone`
No error.

`serErrBadPort`
This port doesn't exist.

`serErrNotOpen`
The port is not open.

`serErrLineErr`
A line error has occurred. Use `SrmGetStatus()` to obtain the exact line error.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmReceive(), SrmReceiveFlush(), SrmReceiveWait()

SrmReceiveFlush Function

Purpose Flushes the receive FIFOs.

Declared In `SerialMgr.h`

Prototype

```
Err SrmReceiveFlush (UInt16 portId,
   Int32 timeout)
```

Parameters

→ *`portId`*
Port ID returned from SrmOpen() or SrmExtOpen().

→ *`timeout`*
Timeout value, in ticks.

Returns This function returns the following error codes:

`errNone`
No error.

`serErrBadPort`
This port doesn't exist.

`serErrNotOpen`
The port is not open.

`serErrNotSupported`
The port is not the foreground port.

`serErrNoDevicesAvail`
No serial devices could be found.

Comments The `timeout` value forces this function to wait a period of ticks after flushing the port to see if more data shows up to be flushed. If more data arrives within the timeout period, the port is flushed again and the timeout counter is reset and waits again. The function only exits after no more bytes are received by the port for the full timeout period since the last flush of the port. To avoid this waiting behavior, specify 0 for the timeout period.

Any errors on the line are cleared before this function returns.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmReceive(), SrmReceiveCheck(), SrmReceiveWait()

SrmReceiveWait Function

Purpose Waits until some number of bytes of data have arrived into the serial receive queue, then returns.

Declared In `SerialMgr.h`

Prototype

```
Err SrmReceiveWait (UInt16 portId, UInt32 bytes,
    Int32 timeout)
```

Parameters

→ *`portID`*
Port ID returned from SrmOpen() or SrmExtOpen().

→ *`bytes`*
Number of bytes to wait for.

→ *`timeout`*
Timeout value, in ticks.

Returns This function returns the following error codes:

`errNone`
No error.

`serErrBadPort`
This port doesn't exist.

`serErrNotOpen`
The port is not open.

`serErrTimeOut`
Unable to receive data within the specified timeout period.

`serErrNotSupported`
The port is not the foreground port.

`serErrBadParam`
The bytes parameter exceeds the size of the receive queue. Use `SrmSetReceiveBuffer()` to increase the size of the receive queue.

`serErrLineErr`
A line error occurred during the receipt of data. Use `SrmGetStatus()` to obtain the exact line error.

`serErrNoDevicesAvail`
No serial devices could be found.

Comments If this function returns no error, the application can either check the number of bytes currently in the receive queue (using `SrmReceiveCheck()`) or it can just specify a buffer and receive the data by calling `SrmReceive()`.

Do not call `SerReceiveWait` from within a wakeup handler. If you do, the `serErrTimeOut` error is returned.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmReceive(), SrmReceiveCheck(), SrmReceiveFlush()

SrmReceiveWindowClose Function

Purpose Closes direct access to the Serial Manager's receive queue.

Declared In `SerialMgr.h`

Prototype

```
Err SrmReceiveWindowClose (UInt16 portId,
   UInt32 bytesPulled)
```

Parameters

→ *portId*
Port ID returned from SrmOpen() or SrmExtOpen().

→ *bytesPulled*
Number of bytes the application read from the receive queue.

Returns This function returns the following error codes:

`errNone`
No error.

`serErrBadPort`
This port doesn't exist.

`serErrNotOpen`
The port is not open.

`serErrNotSupported`
The port is not the foreground port.

`serErrNoDevicesAvail`
No serial devices could be found.

Comments Call this function when the application has read as many bytes as it needs out of the receive queue or it has read all the available bytes.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmReceiveWindowOpen()

SrmReceiveWindowOpen Function

Purpose Provides direct access to the Serial Manager's receive queue.

Declared In `SerialMgr.h`

Prototype

```
Err SrmReceiveWindowOpen (UInt16 portId,
   UInt8 **bufPP, UInt32 *sizeP)
```

Parameters

→ *portId*
Port ID returned from SrmOpen() or SrmExtOpen().

← *bufPP*
Pointer to a pointer to the receive buffer.

← *sizeP*
Available bytes in buffer.

Returns This function returns the following error codes:

`errNone`
No error.

`serErrBadPort`
This port doesn't exist.

`serErrNotOpen`
The port is not open.

`serErrNotSupported`
The port is not the foreground port.

`serErrLineErr`
The data in the queue contains line errors.

`serErrNoDevicesAvail`
No serial devices could be found.

Comments This function lets applications directly access the Serial Manager's receive queue to eliminate buffer copying by the Serial Manager. This access is a "back door" route to the received data. After retrieving data from the buffer, the application must call `SrmReceiveWindowClose()`.

Applications that want to empty the receive buffer entirely should call the `SrmReceiveWindowOpen` and `SrmReceiveWindowClose` functions repeatedly until the buffer size returned is 0.

IMPORTANT: Once an application calls `SrmReceiveWindowOpen`, it should not attempt to receive data via the normal method of calling `SrmReceive()` or `SrmReceiveWait()`, as these functions interfere with direct access to the receive queue.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmReceiveWindowClose()

SrmSend Function

Purpose Sends a block of data out the specified port.

Declared In `SerialMgr.h`

Prototype

```
UInt32 SrmSend (UInt16 portId, const void *bufP,
    UInt32 count, Err *errP)
```

Parameters

→ *portID*
Port ID returned from SrmOpen() or SrmExtOpen().

→ *bufp*
Pointer to data to send.

→ *count*
Length of data buffer, in bytes.

← *errP*
Error code. See the Comments section for details.

Returns Number of bytes of data actually sent.

Comments When `SrmSend` returns, you should check the value returned in the `errP` parameter. If `errNone`, then the entire data buffer was sent. If not `errNone`, then the result equals the number of bytes sent before the error occurred. The possible error values are:

`errNone`
No error.

`serErrBadPort`
This port doesn't exist.

`serErrNotOpen`
The port is not open.

`serErrTimeOut`
Unable to send data within the specified CTS timeout period.

`serErrNoDevicesAvail`
No serial devices could be found.

`serErrConfigurationFailed`
The port configuration has failed.

`serErrNotSupported`
The specified port is not the foreground port.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmSendCheck(), SrmSendFlush(), SrmSendWait()

SrmSendCheck Function

Purpose Checks the transmit FIFO and returns the number of bytes left to be sent.

Declared In `SerialMgr.h`

Prototype

```
Err SrmSendCheck (UInt16 portId,
   UInt32 *numBytesP)
```

Parameters

→ *`portID`*
Port ID returned from SrmOpen() or SrmExtOpen().

← *`numBytesP`*
Number of bytes left in the FIFO queue.

Returns This function returns the following error codes:

`errNone`
No error.

`serErrBadPort`
This port doesn't exist.

`serErrNotOpen`
The port is not open.

`serErrNotSupported`
This feature not supported by the hardware.

`serErrNoDevicesAvail`
No serial devices could be found.

Comments Not all serial devices support this feature.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmSend(), SrmSendFlush(), SrmSendWait()

SrmSendFlush Function

Purpose Flushes the transmit FIFO.

Declared In `SerialMgr.h`

Prototype

```
Err SrmSendFlush (UInt16 portId)
```

Parameters

→ *`portID`*
Port ID returned from SrmOpen() or SrmExtOpen().

Returns This function returns the following error codes:

`errNone`
No error.

`serErrBadPort`
This port doesn't exist.

`serErrNotOpen`
The port is not open.

`serErrNotSupported`
The port is not the foreground port.

`serErrNoDevicesAvail`
No serial devices could be found.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmSend(), SrmSendCheck(), SrmSendWait()

SrmSendWait Function

Purpose Waits until all previous data has been sent from the transmit FIFO, then returns.

Declared In `SerialMgr.h`

Prototype

```
Err SrmSendWait (UInt16 portId)
```

Parameters

→ *portID*
Port ID returned from SrmOpen() or SrmExtOpen().

Returns This function returns the following error codes:

`errNone`
No error.

`serErrBadPort`
This port doesn't exist.

`serErrNotOpen`
The port is not open.

`serErrTimeOut`
Unable to send data within the CTS timeout period.

`serErrNotSupported`
The port is not the foreground port.

`serErrNoDevicesAvail`
No serial devices could be found.

Comments Consider calling this function if your software needs to detect when all data has been transmitted by `SrmSend()`. The `SrmSend` function blocks until all data has been transmitted or a timeout occurs. A subsequent call to `SrmSendWait` blocks until all data queued up for transmission has been transmitted or until another CTS timeout occurs (if CTS handshaking is enabled).

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmSend(), SrmSendCheck(), SrmSendFlush()

SrmSetReceiveBuffer Function

Purpose Installs a new buffer into the Serial Manager's receive queue.

Declared In `SerialMgr.h`

Prototype

```
Err SrmSetReceiveBuffer (UInt16 portId,
   void *bufP, UInt16 bufSize)
```

Parameters

→ *portID*
Port ID returned from SrmOpen() or SrmExtOpen().

→ *bufP*
Pointer to new receive buffer. Ignored if `bufSize` is `NULL`.

→ *bufSize*
Size of new receive buffer in bytes. To remove this buffer and allocate a new default buffer (512 bytes), specify `NULL`.

Returns This function returns the following error codes:

`errNone`
No error.

`serErrBadPort`
This port doesn't exist.

`serErrNotOpen`
This port is not open.

`memErrNotEnoughSpace`
Not enough memory to allocate default buffer.

`serErrNoDevicesAvail`
No serial devices could be found.

Comments The buffer that you pass to this function must remain allocated while you have the serial port open. Before you close the serial port, you must restore the default queue by calling `SrmSetReceiveBuffer` with `NULL` as the `bufP` and `bufSize` arguments.

IMPORTANT: Applications must install the default buffer before closing the port (or disposing of the new receive queue).

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

SrmSetWakeupHandler Function

Purpose Installs a wakeup handler.

Declared In `SerialMgr.h`

Prototype

```
Err SrmSetWakeupHandler (UInt16 portId,
   WakeupHandlerProcPtr procP, UInt32 refCon)
```

Parameters

→ *portID*
Port ID returned from SrmOpen() or SrmExtOpen().

→ *procP*
Pointer to a `WakeupHandlerProcPtr()` function. Specify `NULL` to remove a handler.

→ *refCon*
User-defined data that is passed to the wakeup handler function. This can be a pointer or not.

Returns This function returns the following error codes:

`errNone`
No error.

`serErrBadPort`
This port doesn't exist.

`serErrNotOpen`
The port is not open.

`serErrNoDevicesAvail`
No serial devices could be found.

Comments The wakeup handler is a function in your application that you want to be called whenever there is data ready to be received on the specified port.

The wakeup handler function will not become active until it is primed with a number of bytes that is greater than 0, by the `SrmPrimeWakeupHandler()` function. Every time a wakeup handler is called, it must be re-primed (using `SrmPrimeWakeupHandler`) in order to be called again.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmPrimeWakeupHandler(), WakeupHandlerProcPtr()

Serial Manager Application-Defined Functions

WakeupHandlerProcPtr Function

Purpose Called after some number of bytes are received by the Serial Manager's interrupt function.

Declared In `SerialMgr.h`

Prototype

```
void (*WakeupHandlerProcPtr)(UInt32 refCon)
```

Parameters

→ *refCon*
User-defined data passed from the `SrmSetWakeupHandler()` function.

Returns Returns nothing.

Comments This handler function is installed by calling `SrmSetWakeupHandler()`. The number of bytes after which it is called is specified by `SrmPrimeWakeupHandler()`.

IMPORTANT: Because wakeup handlers are called during interrupt time, they cannot call *any* Palm OS® system functions, including `SrmReceive()`, that may block the system in any way. Wakeup handlers should also be very short so as to reduce interrupt latency.

Two common implementations of wakeup handlers include:

- Calling `EvtWakeup()`, which causes any pending `EvtGetEvent` call to return and then sends a `nilEvent` to the current application.
- Using `SrmReceiveWindowOpen()` and `SrmReceiveWindowClose()` to gain direct access to the receive queue without blocking.

Compatibility Implemented only if New Serial Manager Feature Set Version 1 is present.

See Also SrmPrimeWakeupHandler(), SrmSetWakeupHandler()

71

Old Serial Manager

This chapter provides reference material for the serial manager API:

- Serial Manager Data Structures
- Serial Manager Functions

The header file `SerialMgrOld.h` declares the serial manager API. For more information on the serial manager, see the chapter "Serial Communication" in the *Palm OS Programmer's Companion*, vol. II, *Communications*.

NOTE: The API described in this chapter is obsolete if the New Serial Manager Feature Set is present. The API is still supported for backward compatibility; however, the Serial Manager APIs are preferred.

Serial Manager Data Structures

SerCtlEnum Enum

To perform a control function, applications call `SerControl()`, which performs one of the control operations specified by `SerCtlEnum`, which has the following elements:

Element	Description
`serCtlFirstReserved` = 0	Reserve 0
`serCtlStartBreak`	Turn RS232 break signal on. Applications have to make sure that the break is set long enough to generate a value BREAK! `valueP = 0; valueLenP = 0`

Element	Description
`serCtlStopBreak`	Turn RS232 break signal off: `valueP = 0; valueLenP = 0`
`serCtlBreakStatus`	Get RS232 break signal status (on or off): `valueP` = ptr to Word for returning status (0 = off, !0 = on) `*valueLenP = sizeof(Word)`
`serCtlStartLocalLoopback`	Start local loopback test; `valueP = 0, valueLenP = 0`
`serCtlStopLocalLoopback`	Stop local loopback test `valueP = 0, valueLenP = 0`
`serCtlMaxBaud`	`valueP = ptr` to for returned baud `*valueLenP = sizeof(DWord)`
`serCtlHandshakeThreshold`	Retrieve HW handshake threshold; this is the maximum baud rate that does not require hardware handshaking `valueP` = `ptr` to DWord for returned baud `*valueLenP = sizeof(DWord)`
`serCtlEmuSetBlockingHook`	Set a blocking hook routine. **WARNING!** WARNING: For use with the Simulator on Mac OS only: NOT SUPPORTED ON THE PALM DEVICE. `valueP = ` ptr to SerCallbackEntryType `*valueLenP=sizeof(SerCallbackEntryType)` Returns the old settings in the first argument.
`serCtlLAST`	Add new address entries before this one.

SerSettingsType Struct

Purpose The `SerSettingsType` structure defines serial port attributes; it is used by the calls `SerGetSettings()` and `SerSetSettings()`. The `SerSettingsPtr` type points to a `SerSettingsType` structure.

Prototype

```
typedef struct SerSettingsType {
  UInt32 baudRate;
  UInt32 flags;
  Int32 ctsTimeout;
  } SerSettingsType;
typedef SerSettingsType *SerSettingsPtr;
```

Fields

`baudRate`
: Baud rate

`flags`
: Miscellaneous settings

`ctsTimeout`
: Maximum number of ticks to wait for CTS to become asserted before transmitting; used only when configured with the `serSettingsFlagCTSAutoM` flag.

Serial Manager Functions

SerClearErr Function

Purpose Reset the serial port's line error status.

Declared In `SerialMgrOld.h`

Prototype `Err SerClearErr (UInt16 refNum)`

Parameters

→ *`refNum`*
: The serial library reference number.

Returns

`0`
: No error.

Comments Call `SerClearErr` only after a serial manager function (`SerReceive`, `SerReceiveCheck`, `SerSend`, etc.) returns with the error code `serErrLineErr`.

The reason for this is that `SerClearErr` resets the serial port. So, if `SerClearErr` is called unconditionally while a byte is coming into the serial port, that byte is guaranteed to become corrupted.

The right strategy is to always check the error code returned by a serial manager function. If it 's `serErrLineErr`, call `SerClearErr` immediately. However, don't make unsolicited calls to `SerClearErr`.

When you get `serErrLineErr`, consider flushing the receive queue for a fraction of a second by calling `SerReceiveFlush`. `SerReceiveFlush` calls `SerClearErr` for you.

SerClose Function

Purpose Release the serial port previously acquired by `SerOpen`.

Declared In `SerialMgrOld.h`

Prototype

```
Err SerClose (UInt16 refNum)
```

Parameters

→ *refNum*
: Serial library reference number.

Returns

`0`
: No error.

`serErrNotOpen`
: Port wasn't open.

`serErrStillOpen`
: Port still held open by another process.

Comments Releases the serial port and shuts down serial port hardware if the open count has reached 0. Open serial ports consume more energy from the device's batteries; it's therefore essential to keep a port open only as long as necessary.

Caveat Don't call `SerClose` unless the return value from `SerOpen()` was 0 (zero) or `serErrAlreadyOpen`.

See Also SerOpen()

SerControl Function

Purpose Perform a control function.

Declared In `SerialMgrOld.h`

Prototype

```
Err SerControl (UInt16 refNum, UInt16 op,
   void *valueP, UInt16 *valueLenP)
```

Parameters

→ *refNum*
Reference number of library.

→ *op*
Control operation to perform (`SerCtlEnum`).

↔ *valueP*
Pointer to value for operation.

↔ *valueLenP*
Pointer to size of value.

Returns

`0`
No error.

`serErrBadParam`
Invalid parameter (unknown).

`serErrNotOpen`
Library not open.

Comments This function provides extensible control features for the serial manager. You can

- Turn on/off the RS232 break signal and check its status.
- Perform a local loopback test.
- Get the maximum supported baud rate.
- Get the hardware handshake threshold baud rate.

Compatibility Implemented only if 2.0 New Feature Set is present.

SerGetSettings Function

Purpose Fill in the `SerSettingsType` structure with current serial port attributes.

Declared In `SerialMgrOld.h`

Prototype

```
Err SerGetSettings (UInt16 refNum,
    SerSettingsPtr settingsP)
```

Parameters

→ *refNum*
Serial library reference number.

↔ *settingsP*
Pointer to `SerSettingsType` structure to be filled in.

Returns

`0`
No error.

`serErrNotOpen`
The port wasn't open.

Comments The information returned by this call includes the current baud rate, CTS timeout, handshaking options, and data format options.

See the `SerSettingsType` structure for more details.

See Also SerSend()

SerGetStatus Function

Purpose Return the pending line error status for errors that have been detected since the last time `SerClearErr()` was called.

Declared In `SerialMgrOld.h`

Prototype

```
UInt16 SerGetStatus (UInt16 refNum,
    Boolean *ctsOnP, Boolean *dsrOnP)
```

Parameters

→ *refNum*
Serial library reference number.

→ *ctsOnP*
Pointer to location for storing a Boolean value.

→ *dsrOnP*
Pointer to location for storing a Boolean value.

Returns Returns any combination of the following constants, bitwise OR'ed together:

`serLineErrorParity`
: Parity error.

`serLineErrorHWOverrun`
: Hardware overrun.

`serLineErrorFraming`
: Framing error.

`serLineErrorBreak`
: Break signal detected.

`serLineErrorHShake`
: Line handshake error.

`serLineErrorSWOverrun`
: Software overrun.

Comments When another serial manager function returns an error code of `serErrLineErr`, `SerGetStatus` can be used to find out the specific nature of the line error(s).

The values returned via `ctsOnP` and `dsrOnP` are not meaningful in the present version of the software

See Also SerClearErr()

SerOpen Function

Purpose Acquire and open a serial port with given baud rate and default settings.

Declared In `SerialMgrOld.h`

Prototype

```
Err SerOpen (UInt16 refNum, UInt16 port,
   UInt32 baud)
```

Parameters

→ *refNum*
: Serial library reference number.

→ *port*
: Port number.

→ *baud*
: Baud rate.

Returns

`0`
No error.

`serErrAlreadyOpen`
Port was open. Enables port sharing by "friendly" clients (not recommended).

`serErrBadParam`
Invalid parameter.

`memErrNotEnoughSpace`
Insufficient memory.

Comments

Acquires the serial port, powers it up, and prepares it for operation. To obtain the serial library reference number, call `SysLibFind()` with "Serial Library" as the library name. This reference number must be passed as a parameter to all serial manager functions. The device currently contains only one serial port with port number 0 (zero).

The baud rate is an integral baud value (for example - 300, 1200, 2400, 9600, 19200, 38400, 57600, etc.). The Palm OS® device has been tested at the standard baud rates in the range of 300 - 57600 baud. Baud rates through 1 Mbit are theoretically possible. Use CTS handshaking at baud rates above 19200 (see `SerSetSettings()`).

An error code of 0 (zero) or `serErrAlreadyOpen` indicates that the port was successfully opened. If the port is already open when `SerOpen` is called, the port's open count is incremented and an error code of `serErrAlreadyOpen` is returned. This ability to open the serial port multiple times allows cooperating tasks to share the serial port. Other tasks must refrain from using the port if `serErrAlreadyOpen` is returned and close it by calling `SerClose()`.

SerReceive Function

Purpose Receives `size` bytes worth of data or returns with error if a line error or timeout is encountered.

Declared In `SerialMgrOld.h`

Prototype

```
UInt32 SerReceive (UInt16 refNum, void *bufP,
   UInt32 count, Int32 timeout, Err *errP)
```

Parameters

`refNum`
Serial library reference number.

↔ *`bufP`*
Buffer for receiving data.

→ *`count`*
Number of bytes to receive.

→ *`timeout`*
Interbyte timeout in ticks, 0 for none, -1 forever.

↔ *`errP`*
For returning error code.

Returns Number of bytes received:

`*errP = 0`
No error.

`serErrLineErr`
RS232 line error.

`serErrTimeOut`
Interbyte timeout.

Compatibility Implemented only if 2.0 New Feature Set is present.

NOTE: The old versions of `SerSend` and `SerReceive` are still available as `SerSend10` and `SerReceive10` (not V10).

See Also SerReceive10()

SerReceive10 Function

Purpose Receive a stream of bytes.

Declared In `SerialMgrOld.h`

Prototype

```
Err SerReceive10 (UInt16 refNum, void *bufP,
    UInt32 bytes, Int32 timeout)
```

Parameters

→ *refNum*
The serial library reference number.

→ *bufP*
Pointer to the buffer for receiving data.

→ *bytes*
Number of bytes desired.

→ *timeout*
Interbyte time out in system ticks (-1 = forever).

Returns

`0`
No error. Requested number of bytes was received.

`serErrTimeOut`
Interbyte time out exceeded while waiting for the next byte to arrive.

`serErrLineErr`
Line error occurred (see `SerClearErr()` and `SerGetStatus()`).

Comments `SerReceive` blocks until all the requested data has been received or an error occurs. Because this call returns immediately without any data if line errors are pending, it is important to acknowledge the detection of line errors by calling `SerClearErr()`. If you just need to retrieve all or some of the bytes which are already in the receive queue, call `SerReceiveCheck()` first to get the count of bytes presently in the receive queue.

Compatibility This function corresponds to the 1.0 version of `SerReceive`.

SerReceiveCheck Function

Purpose Return the count of bytes presently in the receive queue.

Declared In `SerialMgrOld.h`

Prototype

```
Err SerReceiveCheck (UInt16 refNum,
   UInt32 *numBytesP)
```

Parameters

→ *refNum*
Serial library reference number.

↔ *numBytesP*
Pointer to location for returning the byte count.

Returns

`0`
No error.

`serErrLineErr`
Line error pending (see `SerClearErr()` and `SerGetStatus()`).

Comments Because this call does not return the byte count if line errors are pending, it is important to acknowledge the detection of line errors by calling `SerClearErr()`.

See Also SerReceiveWait()

SerReceiveFlush Function

Purpose Discard all data presently in the receive queue and flush bytes coming into the serial port. Clear the saved error status.

Declared In `SerialMgrOld.h`

Prototype

```
void SerReceiveFlush (UInt16 refNum,
   Int32 timeout)
```

Parameters

→ *refNum*
Serial library reference number.

→ *timeout*
Interbyte time out in system ticks (-1 = forever).

Returns Returns nothing.

Comments `SerReceiveFlush` blocks until a timeout occurs while waiting for the next byte to arrive.

SerReceiveWait Function

Purpose Wait for at least `bytes` bytes of data to accumulate in the receive queue.

Declared In `SerialMgrOld.h`

Prototype

```
Err SerReceiveWait (UInt16 refNum, UInt32 bytes,
    Int32 timeout)
```

Parameters

→ *`refNum`*
Serial library reference number.

→ *`bytes`*
Number of bytes desired.

→ *`timeout`*
Interbyte timeout in system ticks (-1 = forever).

Returns

`0`
No error.

`serErrTimeOut`
Interbyte timeout exceeded while waiting for next byte to arrive.

`serErrLineErr`
Line error occurred (see `SerClearErr()` and `SerGetStatus()`).

Comments This is the preferred method of waiting for serial input, since it blocks the current task and allows switching the processor into a more energy-efficient state.

`SerReceiveWait` blocks until the desired number of bytes accumulate in the receive queue or an error occurs. The desired number of bytes must be less than the current receive queue size. The default queue size is 512 bytes. Because this call returns immediately if line errors are pending, it is important to acknowledge the detection of line errors by calling `SerClearErr()`.

See Also SerReceiveCheck(), SerSetReceiveBuffer()

SerSend Function

Purpose Send one or more bytes of data over the serial port.

Declared In `SerialMgrOld.h`

Prototype

```
UInt32 SerSend (UInt16 refNum, void *bufP,
   UInt32 count, Err *errP
```

Parameters

→ *refNum*
Serial library reference number.

→ *bufP*
Pointer to data to send.

→ *count*
Number of bytes to send.

↔ *errP*
For returning error code.

Returns Returns the number of bytes transferred.

Stores in `errP`:

`0`
No error.

`serErrTimeOut`
Handshake timeout.

The old calls worked, but they did not return enough info when they failed. The new calls (available in Palm OS v2.0 and greater) add more parameters to solve this problem and make serial communications programming simpler.

Don't call the new functions when running on Palm OS 1.0.

Compatibility Implemented only if 2.0 New Feature Set is present.

NOTE: The old versions of `SerSend` and `SerReceive` are still available as `SerSend10` and `SerReceive10` (not V10).

See Also `SerSend10()`, `SerSendWait()`

SerSend10 Function

Purpose Send a stream of bytes to the serial port.

Declared In `SerialMgrOld.h`

Prototype

```
Err SerSend10 (UInt16 refNum, void *bufP,
    UInt32 size)
```

Parameters

→ *refNum*
Serial library reference number.

→ *bufP*
Pointer to the data to send.

→ *size*
Size (in number of bytes) of the data to send.

Returns

`0`
No error.

`serErrTimeOut`
Handshake timeout (such as waiting for CTS to become asserted).

Comments In the present implementation, `SerSend10` blocks until all data is transferred to the UART or a timeout error (if CTS handshaking is enabled) occurs. Future implementations may queue up the request and return immediately, performing transmission in the background. If your software needs to detect when all data has been transmitted, see `SerSendWait`.

This routine observes the current CTS time out setting if CTS handshaking is enabled (see `SerGetSettings()` and `SerSend()`).

Compatibility This function corresponds to the 1.0 version of `SerSend`.

See Also `SerSend()`, `SerSendWait()`

SerSendFlush Function

Purpose Discard all data presently in the transmit queue.

Declared In `SerialMgrOld.h`

Prototype

```
Err SerSendFlush (UInt16 refNum)
```

Parameters

→ *refNum*
Serial library reference number.

Returns

`0`
No error.

See Also SerSend(), SerSendWait()

SerSendWait Function

Purpose Wait until the serial transmit buffer empties.

Declared In `SerialMgrOld.h`

Prototype

```
Err SerSendWait (UInt16 refNum, Int32 timeout)
```

Parameters

→ *refNum*
Serial library reference number.

→ *timeout*
Reserved for future enhancements. Set to (-1) for compatibility.

Returns

`0`
No error.

`serErrTimeOut`
Handshake timeout (such as waiting for CTS to become asserted).

Comments `SerSendWait` blocks until all data is transferred or a timeout error (if CTS handshaking is enabled) occurs. This routine observes the current CTS timeout setting if CTS handshaking is enabled (see `SerGetSettings()` and `SerSend()`).

See Also SerSend()

SerSetReceiveBuffer Function

Purpose Replace the default receive queue. To restore the original buffer, pass `bufSize` = 0.

Declared In `SerialMgrOld.h`

Prototype

```
Err SerSetReceiveBuffer (UInt16 refNum,
    void *bufP, UInt16 bufSize)
```

Parameters

→ *refNum*
Serial library reference number.

→ *bufP*
Pointer to buffer to be used as the new receive queue.

→ *bufSize*
Size of buffer, or 0 to restore the default receive queue.

Returns Returns 0 if successful.

Comments The specified buffer needs to contain 32 extra bytes for serial manager overhead (its size should be your application's requirement plus 32 bytes). The default receive queue must be restored before the serial port is closed. To restore the default receive queue, call `SerSetReceiveBuffer()` passing 0 (zero) for the buffer size. The serial manager does not free the custom receive queue.

SerSetSettings Function

Purpose Set the serial port settings; that is, change its attributes.

Declared In `SerialMgrOld.h`

Prototype

```
Err SerSetSettings (UInt16 refNum,
    SerSettingsPtr settingsP)
```

Parameters

→ *refNum*
Serial library reference number.

↔ *settingsP*
Pointer to the filled in `SerSettingsType` structure.

Returns

`0`
No error.

`serErrNotOpen`
The port wasn't open.

`serErrBadParam`
Invalid parameter.

Comments The attributes set by this call include the current baud rate, CTS timeout, handshaking options, and data format options. See the definition of the `SerSettingsType` structure for more details.

To do 7E1 transmission, `OR` together:

```
serSettingsFlagBitsPerChar7 |
serSettingsFlagParityOnM |
serSettingsFlagParityEvenM |
serSettingsFlagStopBits1
```

If you're trying to communicate at speeds greater than 19.2 Kbps, you need to use hardware handshaking: `serSettingsFlagRTSAutoM | serSettingsFlagCTSAutoM`.

See Also SerGetSettings()

72

Serial Link Manager

This chapter provides reference material for the serial link manager API. The header file `SerialLinkMgr.h` declares the serial link manager API. For more information on the serial link manager, see the chapter "Serial Communication" in the *Palm OS Programmer's Companion*, vol. II, *Communications*.

Serial Link Manager Functions

SlkClose Function

Purpose Close down the serial link manager.

Declared In `SerialLinkMgr.h`

Prototype

```
Err SlkClose (void)
```

Parameters None.

Returns

`0`
: No error.

`slkErrNotOpen`
: The serial link manager was not open.

Comments When the open count reaches zero, this routine frees resources allocated by serial link manager.

SlkCloseSocket Function

Purpose Closes a socket previously opened with `SlkOpenSocket()`.

The caller is responsible for closing the communications library used by this socket, if necessary.

Declared In `SerialLinkMgr.h`

Prototype

```
Err SlkCloseSocket (UInt16 socket)
```

Parameters

`socket`
The socket ID to close.

Returns

`0`
No error.

`slkErrSocketNotOpen`
The socket was not open.

Comments `SlkCloseSocket` frees system resources the serial link manager allocated for the socket. It does not free resources allocated and passed by the client, such as the buffers passed to `SlkSetSocketListener()`; this is the client's responsibility. The caller is also responsible for closing the communications library used by this socket.

See Also `SlkOpenSocket()`

SlkFlushSocket Function

Purpose Flush the receive queue of the communications library associated with the given socket.

Declared In `SerialLinkMgr.h`

Prototype

```
Err SlkFlushSocket (UInt16 socket, Int32 timeout)
```

Parameters

→ *`socket`*
Socket ID.

→ *`timeout`*
Interbyte timeout in system ticks.

Returns

`0`
No error.

`slkErrSocketNotOpen`
The socket wasn't open.

SlkOpen Function

Purpose Initialize the serial link manager.

Declared In `SerialLinkMgr.h`

Prototype
```
Err SlkOpen (void)
```

Parameters None.

Returns

`0`
No error.

`slkErrAlreadyOpen`
No error.

Comments Initializes the serial link manager, allocating necessary resources. Return codes of 0 (zero) and `slkErrAlreadyOpen` both indicate success. Any other return code indicates failure. The `slkErrAlreadyOpen` function informs the client that someone else is also using the serial link manager. If the serial link manager was successfully opened by the client, the client needs to call `SlkClose()` when it finishes using the serial link manager.

SlkOpenSocket Function

Purpose Open a serial link socket and associate it with a communications library. The socket may be a known static socket or a dynamically assigned socket.

Declared In `SerialLinkMgr.h`

Prototype
```
Err SlkOpenSocket (UInt16 portID, UInt16 *socketP,
   Boolean staticSocket)
```

Parameters

`portID`
Comm library reference number for socket.

`socketP`
Pointer to location for returning the socket ID.

`staticSocket`
If `TRUE`, `*socketP` contains the desired static socket number to open. If `FALSE`, any free socket number is assigned dynamically and opened.

Returns

`0`
No error.

`slkErrOutOfSockets`
No more sockets can be opened.

Comments

The communications library must already be initialized and opened (see `SerOpen()`). When finished using the socket, the caller must call `SlkCloseSocket()` to free system resources allocated for the socket. For information about well-known static socket IDs, see The Serial Link Protocol.

SlkReceivePacket Function

Purpose

Receive and validate a packet for a particular socket or for any socket. Check for format and checksum errors.

Declared In

`SerialLinkMgr.h`

Prototype

```
Err SlkReceivePacket (UInt16 socket,
   Boolean andOtherSockets,
   SlkPktHeaderPtr headerP, void *bodyP,
   UInt16 bodySize, Int32 timeout)
```

Parameters

→ *socket*
The socket ID.

→ *andOtherSockets*
If `TRUE`, ignore destination in packet header.

↔ *headerP*
Pointer to the packet header buffer (size of SlkPktHeaderType).

↔ *bodyP*
Pointer to the packet client data buffer.

→ *bodySize*
Size of the client data buffer (maximum client data size which can be accommodated).

→ *timeout*
Maximum number of system ticks to wait for beginning of a packet; -1 means wait forever.

Returns

`0`
No error.

`slkErrSocketNotOpen`
The socket was not open.

`slkErrTimeOut`
Timed out waiting for a packet.

`slkErrWrongDestSocket`
The packet being received had an unexpected destination.

`slkErrChecksum`
Invalid header checksum or packet CRC-16.

`slkErrBuffer`
Client data buffer was too small for packet's client data.

If `andOtherSockets` is `FALSE`, this routine returns with an error code unless it gets a packet for the specific socket.

If `andOtherSockets` is `TRUE`, this routine returns successfully if it sees any incoming packet from the communications library used by `socket`.

Comments

You may request to receive a packet for the passed socket ID only, or for any open socket which does not have a socket listener. The parameters also specify buffers for the packet header and client data, and a timeout. The timeout indicates how long the receiver should wait for a packet to begin arriving before timing out. If a packet is received for a socket with a registered socket listener, it will be dispatched via its socket listener procedure. On success, the packet header buffer and packet client data buffer is filled in with the actual size of the packet's client data in the packet header's `bodySize` field.

SlkSendPacket Function

Purpose Send a serial link packet via the serial output driver.

Declared In `SerialLinkMgr.h`

Prototype

```
Err SlkSendPacket (SlkPktHeaderPtr headerP,
    SlkWriteDataPtr writeList)
```

Parameters

↔ *headerP*
Pointer to the packet header structure with client information filled in (see Comments).

→ *writeList*
List of packet client data blocks (see Comments).

Returns

`0`
No error.

`slkErrSocketNotOpen`
The socket was not open.

`slkErrTimeOut`
Handshake timeout.

Comments `SlkSendPacket` stuffs the signature, client data size, and the checksum fields of the packet header. The caller must fill in all other packet header fields. If the transaction ID field is set to 0 (zero), the serial link manager automatically generates and stuffs a new non-zero transaction ID. The array of `SlkWriteDataType` structures enables the caller to specify the client data part of the packet as a list of noncontiguous blocks. The end of list is indicated by an array element with the `size` field set to 0 (zero). This call blocks until the entire packet is sent out or until an error occurs.

SlkSetSocketListener Function

Purpose Register a socket listener for a particular socket.

Declared In `SerialLinkMgr.h`

Prototype

```
Err SlkSetSocketListener (UInt16 socket,
    SlkSocketListenPtr socketP)
```

Parameters

→ *socket*
Socket ID.

→ *socketP*
Pointer to a SlkSocketListenType structure.

Returns

`0`
No error.

`slkErrBadParam`
Invalid parameter.

`slkErrSocketNotOpen`
The socket was not open.

Comments

Called by applications to set up a socket listener.

Since the serial link manager does not make a copy of the `SlkSocketListenType` structure, but instead saves the passed pointer to it, the structure

- must **not** be an automatic variable (that is, local variable allocated on the stack)
- may be a global variable in an application
- may be a locked chunk allocated from the dynamic heap

The `SlkSocketListenType` structure specifies pointers to the socket listener procedure and the data buffers for dispatching packets destined for this socket. Pointers to two buffers must be specified: the packet header buffer (size of `SlkPktHeaderType`), and the packet body (client data) buffer. The packet body buffer must be large enough for the largest expected client data size. Both buffers may be application global variables or locked chunks allocated from the dynamic heap.

The socket listener procedure is called when a valid packet is received for the socket. Pointers to the packet header buffer and the packet body buffer are passed as parameters to the socket listener procedure.

NOTE: The application is responsible for freeing the `SlkSocketListenType` structure or the allocated buffers when the socket is closed. The serial link manager doesn't do it.

Compatibility

If either 5.0 New Feature Set or Palm OS Cobalt Feature Set is present this function is unimplemented.

SlkSocketPortID Function

Purpose Get the port ID associated with a particular socket; for use with the new serial manager.

Declared In `SerialLinkMgr.h`

Prototype

```
ErrSlkSocketPortID (UInt16 socket,
    UInt16 *portIDP)
```

Parameters

→ *socket*
The socket ID.

↔ *portIDP*
Pointer to location for returning the port ID.

Returns

`0`
No error.

`slkErrSocketNotOpen`
The socket was not open.

Compatibility Implemented only if New Serial Manager Feature Set is present.

SlkSocketSetTimeout Function

Purpose Set the interbyte packet receive-timeout for a particular socket.

Declared In `SerialLinkMgr.h`

Prototype

```
Err SlkSocketSetTimeout (UInt16 socket,
    Int32 timeout)
```

Parameters

→ *socket*
Socket ID.

→ *timeout*
Interbyte packet receive-timeout in system ticks.

Returns

`0`
No error.

`slkErrSocketNotOpen`
The socket was not open.

73

Telephony Basic Services

This chapter provides reference material for the Telephony API, which you can use to interface with telephone systems and equipment. This chapter discusses the following topics:

- Telephony Data Structures
- Telephony Constants
- Telephony Functions
- Feature Support Functions

The header file `TelephonyMgr.h` declares the telephony manager API. The header file `TelephonyMgrType.h` declares the data structures that you use with the telephony manager API.

For more information about using the telephony manager, see Chapter 10, "Telephony Manager," in *Palm OS Programmer's Companion*, vol. II, *Communications*.

Telephony Service Types

The telephony API organizes functions within sets called **service sets**. Each service set contains a related set of functions that may or may not be available on a particular mobile device or network. You can use the `TelIs<ServiceSet>Available()` function to determine if a service set is supported in the current environment, and you can use the `TelIs<FunctionName>Supported()` to determine if a specific function is supported in the current environment.

The telephony API documentation has been split into several chapters. Each chapter covers one or more of the service sets, as shown in Table 73.1

Table 73.1 Telephony service types

Service prefix	Functionality	Chapter	Description
`Tel`	Basic	Chapter 73, "Telephony Basic Services."	Basic functions that are always available.
`TelCfg`	Configuration	Chapter 74, "Telephony Security and Configuration."	Allows configuration of the phone, including Short Message Services (SMS) configuration.
`TelDtc`	Data calls	Chapter 76, "Telephony Calls."	Functions for handling data calls.
`TelEmc`	Emergency calls	Chapter 76, "Telephony Calls."	Functions for handling emergency calls.
`TelInf`	Information	Chapter 73, "Telephony Basic Services."	Functions for retrieving information about the phone.
`TelNwk`	Network	Chapter 75, "Telephony Network."	Provides network oriented services, including authorization, signal level, search mode, and related operations.
`TelOem`	OEM	Chapter 73, "Telephony Basic Services."	Provides OEMs with the ability to incorporate custom functionality.
`TelPhb`	Phone book	Chapter 78, "Telephony Phone Book."	Functions for managing the phone book.

Table 73.1 Telephony service types *(continued)*

Service prefix	Functionality	Chapter	Description
`TelPow`	Power	Chapter 73, "Telephony Basic Services."	Provides access to power supply level.
`TelSms`	Short Message Service	Chapter 77, "Telephony SMS."	Addresses the SMS, including functions for reading, sending, replying to, and deleting short messages.
`TelSnd`	Sound	Chapter 73, "Telephony Basic Services."	Functions for playing key tones on and muting the phone.
`TelSpc`	Speech calls	Chapter 76, "Telephony Calls."	Function for handling voice calls, including dual tone modulated frequency (DTMF) sounds.
`TelSty`	Security	Chapter 74, "Telephony Security and Configuration."	Supports PIN code management for access to phone and Subscriber Identity Module (SIM) security-related features.

Telephony Data Structures

This section describes the data structures used with the basic services portion of the telephony API.

TelAppID Typedef

Purpose The type for a telephone application attachment identifier. The application attachment identifier uniquely identifies your

application. You obtain this value from the `TelOpen()` and pass it to other Telephony Manager calls.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef UInt32 TelAppID
```

TelEventType Struct

Purpose The `TelGetEvent()` and `TelGetTelephonyEvent()` functions both return a `TelEventType` structure to provide information about a telephony-related event.

You call the `TelGetEvent()` function to retrieve telephony and other events.

You call the `TelGetTelephonyEvent()` function to retrieve only telephony events. This function does not consume non-telephony events.

Note that the first five fields of the `TelEventType` structure are the same as the first five fields of the `EventType` structure, which is described in Chapter 2, "Palm OS Events."

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelEventType {
   eventsEnum eType;
   Boolean penDown;
   UInt8 tapCount;
   Int16 screenX;
   Int16 screenY;
   UInt16 functionId;
   UInt16 transId;
   MemPtr *paramP;
   Err returnCode;
} TelEventType
```

Fields

`eType`

`kTelTelephonyEvent` to specify that a telephony event has occurred. `nilEvent` to specify that no telephy event has occurred.

`penDown`
: `true` if the pen was down at the time of the event, and `false` if the pen was up.

  Note that this field is not filled in for telephony events.

`tapCount`
: The number of taps received at this location.

  Note that this field is not filled in for telephony events.

`screenX`
: Window-relative position of the pen in standard coordinates (number of coordinates from the left bound of the window).

  Note that this field is not filled in for telephony events.

`screenY`
: Window-relative position of the pen in standard coordinates (number of coordinates from the top left of the window).

  Note that this field is not filled in for telephony events.

`functionId`
: One of the Message Identifier Constants constants, that identifies the asynchronous function whose completion generated this event.

`transId`
: The transaction ID that was associated with this asynchronous function call.

`paramP`
: A pointer to a parameter structure that was passed when an asynchronous call was made.

`returnCode`
: The return code of the asynchronously called function. The value of this field is `errNone` if the function succeeded, or an error code if the function failed.

TelGetCallStateType Struct

Purpose The `TelGetCallState()` function uses the `TelGetCallStateType` structure to retrieve information about the current state of the connected phone.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelGetCallStateType {
   UInt8 state;
   UInt8 callType;
   UInt8 callServiceType;
   UInt8 numberSize;
   Char *number;
} TelGetCallStateType
```

Fields

← `state`
: Filled in with one of the Telephone Call State Constants, which indicates the current state of the telephone call.

← `callType`
: Filled in with one of the Telephone Call Type Constants, which indicates the call type of the current telephone call.

← `callServiceType`
: Filled in with one of the Telephone Call Service Type Constants, which indicates the call service type of the current telephone call.

↔ `numberSize`
: The size of the `number` string buffer. When the structure is used as an input parameter, this is the allocated size, in bytes, of the buffer.

  Upon return, this is the actual size of the string, including the null terminator character. If the `number` buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the data, and the function using this structure generates a `telErrBufferSize` error.

← `number`
: A buffer into which the telephone number string is stored.

  When the structure describes an incoming telephone call, this is the incoming telephone number. When the structure

describes an outgoing telephone call, this is the telephone number that has been called.

Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a `telErrBufferSize` error.

TelInfGetInformationType Struct

Purpose The `TelInfGetInformation()` function uses the `TelInfGetInformationType` structure to retrieve information about the phone with which you are communicating.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelInfGetInformationType {
   UInt8 infoType;
   UInt16 size;
   Char *value;
} TelInfGetInformationType
```

Fields

→ `infoType`
: The type of information to retrieve. This must be one of the Information Type Constants.

↔ `size`
: The size of the `value` buffer. When the structure is used as an input parameter, this is the allocated size, in bytes, of the buffer.

  Upon return, this is the actual size of the buffer. If the `value` buffer is too small to contain all of the retrieved information, this field is assigned the entire length of the data, and the function using this structure generates a `telErrBufferSize` error.

← `value`
: A buffer into which the information is stored.

TelNotificationType Struct

Purpose Holds information for Telephony Manager notifications.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelNotificationType {
   UInt32 notificationData;
   UInt32 notificationData2;
   UInt32 timeStamp;
   UInt16 notificationId;
   UInt8 priority;
} TelNotificationType
```

Fields

`notificationData`
: Various notification-specific data.

`notificationData2`
: Various notification-specific data.

`timeStamp`
: Time the event occurred, expressed as the number of seconds elapsed since 12:00 A.M. on January 1, 1904.

`notificationId`
: Identifies the type of event that occurred. One of the constants described in "Notification Identifiers" on page 1680.

`priority`
: One of the constants described in "Notification Priorities" on page 1677.

Comments This structure is passed for the value of the `notifyDetailsP` field in the notification parameter block of a `kTelTelephonyNotification`.

TelOemCallType Struct

Purpose You use the `TelOemCallType` to specify a `TelOemCall()` function.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelOemCallType {
   UInt32 OemId;
   UInt8 funcId;
   void *paramP;
} TelOemCallType
```

Fields

→ `OemID`
The unique ID of the OEM function set.

→ `funcID`
The ID of the function within the OEM function set.

↔ `paramP`
A pointer to a parameter block that is passed to the OEM function. The OEM function might modify some of the fields in the parameter block.

TelSendCommandStringType Struct

Purpose The `TelSendCommandString()` function uses the `TelSendCommandStringType` structure to send a command string.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSendCommandStringType {
   Char *commandString;
   Char *resultString;
   UInt16 resultSize;
   UInt32 timeOut;
} TelSendCommandStringType
```

Fields

→ `commandString`
The command string to send.

← `resultString`
The result string.

↔ `resultSize`

The size of the `resultString` string buffer. When the structure is used as an input parameter, this is the allocated size, in bytes, of the buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the `resultString` buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the data, and the function using this structure generates a `telErrBufferSize` error.

→ `timeOut`

The number of milliseconds before timing out.

TelSndPlayKeyToneType Struct

Purpose The `TelSndPlayKeyTone()` function uses the `TelSndPlayKeyToneType` structure to specify a key tone.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSndPlayKeyToneType {
   UInt8 keycode;
   UInt8 type;
} TelSndPlayKeyToneType
```

Fields

→ `keycode`

The keycode of the key tone to play. This should be one of the values declared under "Keycode Constants" on page 1669.

→ `type`

The tone type. This should be one of the following constants:

`0x00`

Play the key sound as a single tone.

`0x01`

Play the key sound as a multiple tones.

Telephony Constants

Battery Status Constants

Purpose The battery status constants provide information about the phone's battery.

Declared In `TelephonyMgr.h`

Constants

`#define kTelPowBatteryPowered 0`
The phone is powered by a battery.

`#define kTelPowBatteryNotPowered 1`
The phone has a battery connected to it, but is not using that battery.

`#define kTelPowNoBattery 2`
The phone does not have a battery connected to it.

`#define kTelPowBatteryFault 3`
The phone has a recognized power fault; calls are currently inhibited.

Keycode Constants

Purpose The keycode constants represent the keys on a telephony keypad.

Declared In The keypad keys are identified by numeric value; there are currently no symbolic constants declared to represent these values.

Constants

`0x00 - 0x09`
The numeric keys 0 through 9 on the phone keypad.

`0x23`
The POUND(#) key on the phone keypad.

`0x2A`
The STAR(*) key on the phone keypad.

`0x45`
The SEND key on the phone keypad.

`0x46`
The END key on the phone keypad.

`0x47`
The CLEAR key on the phone keypad.

`0x48`
The SAVE key on the phone keypad.

Telephone Call State Constants

Purpose The `TelGetCallStateType` structure uses the telephone call state constants to encode the current state of the connected telephone call.

Declared In `TelephonyMgr.h`

Constants

`#define kTelCallIdle 0`
The connection is idle.

`#define kTelCallConnecting 1`
A telephone call is currently connecting.

`#define kTelCallConnected 2`
A telephone call is currently connected.

`#define kTelCallRedial 3`
A telephone call is being re-dialed.

`#define kTelCallIncoming 4`
A telephone call is currently incoming.

`#define kTelCallIncomingAck 5`
An incoming telephone call is currently being acknowledged.

`#define kTelCallDisconnecting 6`
A telephone call is being disconnected.

Telephone Call Type Constants

Purpose The `TelGetCallStateType` structure uses the telephone call type constants to encode the type of the current telephone call.

Declared In `TelephonyMgr.h`

Constants

`#define kTelCallTypeOutgoing 0`
An outgoing telephone call.

`#define kTelCallTypeIncoming 1`
An incoming telephone call.

Telephone Call Service Type Constants

Purpose The `TelGetCallStateType` structure uses the telephone call service type constants to encode the service type of the current telephone call.

Declared In `TelephonyMgr.h`

Constants

`#define kTelCallServiceVoice 0`
A voice telephone call.

`#define kTelCallServiceData 1`
A data telephone call.

Error Code Constants

Purpose The Telephony Manager functions return the error code constants shown in the following table to indicate their status.

Declared In `TelephonyMgr.h`

Constants

`#define telErrorClass (appErrorClass + 0x100)`
Base value for Telephony Manager errors.

`#define telErrBufferSize (telErrorClass | 0x0A)`
One of the buffers used to retrieve data is too small.

`#define telErrCodingScheme (telErrorClass | 0x24)`
The coding scheme specified for the short message is not valid.

`#define telErrCommandFailed (telErrorClass | 0x0E)`
The specified command could not be performed by the phone. Check the phone driver.

`#define telErrEntryNotFound (telErrorClass | 0x1C)`
The specified entry was not found.

`#define telErrFeatureNotSupported (telErrorClass | 0x0B)`
The specified feature is not supported by the phone or network.

`#define telErrGenericDrvNotFound (telErrorClass | 0x2A)`
The generic driver could not be found.

`#define telErrInvalidAppId (telErrorClass | 0x07)`
The specified application ID is not valid.

`#define telErrInvalidDial (telErrorClass | 0x1F)`
The dial string contains an invalid character.

`#define telErrInvalidIndex (telErrorClass | 0x1B)`
The index specified for accessing a value in storage is incorrect.

`#define telErrInvalidParameter (telErrorClass | 0x22)`
A parameter is not valid.

`#define telErrInvalidString (telErrorClass | 0x1E)`
The text string contains an invalid character.

`#define telErrLibStillInUse (telErrorClass | 0x34)`
The shared lib is currently being used by another application. Do not unload it!

`#define telErrLimitedCompatibility (telErrorClass | 0x32)`
Current driver is only partially compatible with the connected phone.

`#define telErrMemAllocation (telErrorClass | 0x03)`
A memory allocation error occurred.

`#define telErrMsgAllocation (telErrorClass | 0x01)`
The telephony messages pool is empty; a message could not be allocated.

`#define telErrNetworkTimeOut (telErrorClass | 0x21)`
The network did not reply within the standard time delay amount.

`#define telErrNoNetwork (telErrorClass | 0x20)`
There is no network available.

`#define telErrNoSIMInserted (telErrorClass | 0x11)`
The SIM card is not inserted.

`#define telErrNoSpecificDrv (telErrorClass | 0x2B)`
A specific driver was not specified.

`#define telErrNotInstalled (telErrorClass | 0x2E)`
The shared library could not be installed.

`#define telErrPassword (telErrorClass | 0x17)`
The password is not correct.

`#define telErrPhoneCodeRequired (telErrorClass | 0x10)`
A phone code is required.

`#define telErrPhoneComm (telErrorClass | 0x0C)`
The communication link with the phone is down.

`#define telErrPhoneMemAllocation (telErrorClass | 0x1A)`
The phone's memory is full.

`#define telErrPhoneMemFailure (telErrorClass | 0x1D)`
The phone encountered a memory error.

`#define telErrPhoneNumber (telErrorClass | 0x25)`
One of the following errors has occurred: the phone number is wrong, the SMS center is not valid, or the receiver phone number is wrong for the SMS.

`#define telErrPhoneReply (telErrorClass | 0x0D)`
The phone reply syntax is incorrect. Check the phone driver.

`#define telErrPhoneToSIMPINRequired (telErrorClass | 0x28)`
A phone 2 SIM PIN code is required

`#define telErrPIN2Required (telErrorClass | 0x18)`
A PIN2 code is required.

`#define telErrPINRequired (telErrorClass | 0x12)`
A PIN code is required.

`#define telErrProfileConflict (telErrorClass | 0x33)`
Current profile conflicts with the requested profile.

`#define telErrPUK2Required (telErrorClass | 0x19)`
A PUK2 code is required.

`#define telErrPUKRequired (telErrorClass | 0x13)`
A PUK code is required.

`#define telErrResultBusyResource (telErrorClass | 0x06)`
A resource is busy.

`#define telErrResultTimeOut (telErrorClass | 0x04)`
A time-out was reached.

`#define telErrResultUserCancel (telErrorClass | 0x05)`
The user cancelled the action.

`#define telErrSecurity (telErrorClass | 0x09)`
Access to the phone was not granted.

`#define telErrSettings (telErrorClass | 0x30)`
The telephony settings are not valid; this is due to 1) the Phone Panel preferences do not exist, or 2) the Telephony Profile is not correctly set.

`#define telErrSIMBusy (telErrorClass | 0x15)`
The SIM could not reply.

`#define telErrSIMFailure (telErrorClass | 0x14)`
The SIM is not working properly.

`#define telErrSIMWrong (telErrorClass | 0x16)`
The SIM is not accepted by the phone.

`#define telErrSpcCallError (telErrorClass | 0x2D)`
The voice telephone call encountered an error.

`#define telErrSpcLineIsBusy (telErrorClass | 0x0F)`
The voice telephone call failed.

`#define telErrSpcLineIsReleased (telErrorClass | 0x2C)`
The voice telephone call has been released.

`#define telErrSpecificDrvNotFound (telErrorClass | 0x29)`
The specified driver could not be found.

`#define telErrTooManyApps (telErrorClass | 0x08)`
The applications table is full.

`#define telErrTTaskNotFound (telErrorClass | 0x35)`
The Telephony Task could not be found.

`#define telErrTTaskNotRunning (telErrorClass | 0x27)`
The Telephony Task is not running.

`#define telErrUnavailableValue (telErrorClass | 0x31)`
The requested value can not be retrieved at the specified time. This is usually due to a `TelSpcGetCallerNumber` request when there is no active line.

`#define telErrUnknown (telErrorClass | 0x02)`
An unknown telephony manager error occurred.

`#define telErrValidityPeriod (telErrorClass | 0x23)`
The validity period specified for the short message is not valid.

`#define telErrValueStale (telErrorClass | 0x26)`
The information could not be retrieved; a copy of the most recently retrieved value has been returned instead.

`#define telErrVersion (telErrorClass | 0x2F)`
The shared library version does not match the version associated with the application.

Information Type Constants

Purpose The `TelInfGetInformationType` structure uses the information type constants to encode the type of information to retrieve about the phone.

Declared In `TelephonyMgr.h`

Constants

`#define kTelInfPhoneBrand 0`
The brand name of the phone.

`#define kTelInfPhoneModel 1`
The model number of the phone.

`#define kTelInfPhoneRevision 2`
The revision number of the phone.

Message Identifier Constants

The message identifier constants are used with asynchronous calls to identify which telephony function is being or has been called. The `TelMessages` enumeration defines a constant for each function name.

Each message identifier constant has the form:

`k`*`functionName`*`Message`

where *functionName* is replaced by a function name.

The following table shows examples of message identifier constants. For a complete list, see the `TelephonyMgrTypes.h` file.

Constant	Function
`kTelGetCallStateMessage`	`TelGetCallState()`
`kTelNwkSelectNetworkMessage`	`TelNwkSelectNetwork()`
`kTelSmsReadMessageMessage`	`TelSmsReadMessage()`

TelServices Enum

Purpose The service set constants specify a set of API services.

Declared In `TelephonyMgrTypes.h`

Constants

`kTelNwkServiceId`
The network service set.

`kTelStyServiceId`
The security service set.

`kTelPowServiceId`
The power service set.

`kTelCfgServiceId`
The configuration service set.

`kTelSmsServiceId`
The short message service set.

`kTelEmcServiceId`
The emergency telephone call service set.

`kTelSpcServiceId`
The speech telephone call service set.

`kTelDtcServiceId`
The data telephone call service set.

`kTelPhbServiceId`
The phone book service set.

`kTelOemServiceId`
The OEM service set.

`kTelSndServiceId`
The sound service set.

`kTelInfServiceId`
The information service set.

Notification Priorities

Purpose Notification priorities used in the `priority` field of the `TelNotificationType` structure.

Declared In `TelephonyMgr.h`

Constants

```
#define kTelCallNotificationPriority 0
```
Voice call.

```
#define kTelSmsNotificationPriority 1
```
SMS message.

Call Classes

Purpose Not currently used.

Declared In `TelephonyMgr.h`

Constants

```
#define kTelSpeechCallClass 0
#define kTelDataCallClass 1
#define kTelFaxCallClass 2
```

Telephony Manager Database

Purpose Identifiers for the Telephony Manager database file.

Declared In `TelephonyMgr.h`

Constants

`#define kTelMgrDatabaseCreator sysFileCTelMgrLib`
The creator ID used for the Telephony Manager library's database file.

`#define kTelMgrDatabaseType sysFileTLibrary`
The type used for the Telephony Manager's database file.

`#define kTelMgrLibName "Telephony Library"`
The name of the Telephony Manager database.

Version Constants

Purpose Version of the Telephony Manager and SMS API.

Declared In `TelephonyMgr.h`

Constants

```
#define kTelMgrVersion
  sysMakeROMVersion(kTelMgrVersionMajor,
  kTelMgrVersionMinor, kTelMgrVersionFix,
  sysROMStageBeta, kTelMgrVersionBuild)
```
The Telephony Manager version information.

`#define kTelMgrVersionMajor 1`
The major version number.

`#define kTelMgrVersionMinor 0`
The minor version number.

`#define kTelMgrVersionFix 3`
The fix number.

`#define kTelMgrVersionBuild 0`
The build number.

`#define kTelSmsAPIVersion 0x0001`
Version of the SMS API.

Comments **IMPORTANT:** The shared library version number differs depending on which version of Palm OS you are targeting. This means that you may need to change how the `kTelMgrVersion` constant is constructed. If you are targeting Palm OS version 4.2, the minor version is 1 and the fix level is 0. For Palm OS Garnet, the minor version number is 0 and the fix level is 3. In `TelMgr.h`, make sure that the appropriate values of `kTelMgrVersionMinor` and `kTelMgrVersionFix` are uncommented, and that the values that don't apply are commented out.

Events and Notifications

Purpose Identifiers for the events and notifications broadcast by the Telephony Manager.

Declared In `TelephonyMgr.h`

Constants

`#define kTelTelephonyEvent 0x1200`

An event posted to the event queue when an asynchronously called Telephony Manager function completes. The `TelEventType` structure contains the information about the event.

`#define kTelTelephonyNotification 'tmgr'`

Identifies a notification broadcast by the Telephony Manager, which occurs whenever telephony events occur. Applications interested in such events can register to receive this notification.

The `notifyDetailsP` field of the notification parameter block points to a `TelNotificationType` structure.

Miscellaneous Constants

Purpose Other constants defined in `TelephonyMgr.h`.

Declared In `TelephonyMgr.h`

Constants

`#define kTelInfiniteDelay 0xFFFFFFFF`

Specifies no time out on certain calls. This value is used internally by the Telephony Manager.

`#define kTelInvalidAppId ((TelAppID)-1)`
You can use this constant to initialize the *telDescP* parameter to the `TelOpen()` and `TelOpenProfile()` functions. Upon return, it contains a valid application identifier.

`#define kTelLocationSeparator ';'`
Used to separate tokens in a location string.

`#define kTelSpcCallingLineId -1`

`#define kTelMaxPhoneNumberLen 30`
The maximum length of a phone number.

Notification Identifiers Enum

Purpose Identifies the type of telephony notification. These values are used in the `id` field of the `TelNotificationType` structure.

Declared In `TelephonyMgr.h`

Constants

`kTelSmsLaunchCmdIncomingMessage`
Incoming SMS message. The actual message can be retrieved with `TelSmsReadMessage()` or `TelSmsReadSubmittedMessages()`.

`kTelSpcLaunchCmdIncomingCall`
Incoming voice call. The call can be answered with `TelSpcAcceptCall`.

`kTelSpcLaunchCmdCallerIDAvailable`
Caller ID on an incoming call is available. The information can be retrieved with the `TelSpcGetCallerNumber()` function.

`kTelSpcLaunchCmdCallReleased`
A voice call has been released.

`kTelSpcLaunchCmdCallBusy`
Broadcast as a result of a `TelSpcCallNumber()` function call if the number that the application is attempting to call is busy.

`kTelSpcLaunchCmdCallConnect`
Broadcast as a result of a `TelSpcCallNumber()` function when the call is connected.

`kTelSpcLaunchCmdCallError`
: Broadcast as a result of a `TelSpcCallNumber()` function if an error occurs or the user cancels the operation before the call can be connected.

`kTelEmcLaunchCmdCallReleased`
: An emergency call has been released.

`kTelEmcLaunchCmdCallBusy`
: Broadcast as a result of the `TelEmcCall()` function if the emergency call gets a busy signal.

`kTelEmcLaunchCmdCallConnect`
: Broadcast as a result of the `TelEmcCall()` function when the call is connected.

`kTelEmcLaunchCmdCallError`
: Broadcast as a result of the `TelEmcCall()` function if an error occurs or the user cancels the operation before the call can be connected.

`kTelLastLaunchCode`
: Specifies the last notification constant in the enumeration.

Telephony Functions

This section describes the functions used with the basic services portion of the telephony API.

TelCancel Function

Purpose Cancels an asynchronous function call.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelCancel(UInt16 iRefnum, TelAppID iAppId,
   UInt16 iTransId, UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

→ *iTransId*
: The transaction ID associated with the function that you are cancelling.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function call was successfully cancelled. Returns the `telErrCommandFailed` error code if the function call could not be cancelled.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` upon success or an error code upon failure.
`transId`	The transaction ID of the operation.
`functionId`	`kTelUrqCancelMessage`
`paramId`	Points to the unsigned integer value passed to this function in the `iTransId` parameter.

This function cancels a pending asynchronous function call. You can cancel any asynchronous call except for an asynchronous call to the `TelCancel` function.

The function call that is cancelled returns the `telErrUserCancel` error code.

Compatibility

Implemented only if 4.0 New Feature Set is present.

TelClose Function

Purpose Close the shared library.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelClose(UInt16 iRefnum, TelAppID iAppId)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

Returns Returns an error code, or error none if the library was successfully closed. If the library is currently being used by another application, this function returns the `telErrLibStillInUse` error code.

Comments Call this function when you are done with the telephony manager. You can only use this function synchronously.

If no other application is using the telephony manager, this function stops the Telephony task and releases any resources used by the telephony manager.

See Also `TelOpen()`

TelClosePhoneConnection Function

Purpose Closes down communications with the connected phone.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelClosePhoneConnection(UInt16 iRefnum,
   TelAppID iAppId, UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` upon success or an error code upon failure.
`transId`	The transaction ID of the operation.
`paramP`	A `NULL` pointer.
`functionId`	`kTelUrqClosePhoneConnectionMessage`

Call this function when you have finished communications with the phone and are ready to disconnect from it.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelOpenPhoneConnection()`

TelGetCallState Function

Purpose

Retrieves the current telephone call state information.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelGetCallState(UInt16 iRefnum,
   TelAppID iAppId,
   TelGetCallStateType *ioParamP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*

The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioParamP*
A pointer to a `TelGetCallStateType` structure that describes the state of the current telephone call.

On input, the `numberSize` field of this structure specifies the allocated size of the `number` buffer. Upon return, the `numberSize` field specifies the actual size of the telephone number, even if it was truncated to fit into the buffer.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` upon success or an error code upon failure.
`transId`	The transaction ID of the operation.
`paramP`	Points to the `TelGetCallStateType` structure passed to this function in the `ioCallState` parameter.
`functionId`	`kTelGetCallStateMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments

This function retrieves information about the current telephone call state of the connection with the phone, and stores that information into the supplied `TelGetCallStateType` structure.

The current incoming or outgoing telephone call number is stored into the `number` field of the `TelGetCallStateType` structure referenced by `ioCallStateP`. If the `number` field buffer is too small to contain the complete telephone number, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `numberSize` field of the structure is always updated to contain the actual size of the complete telephone number.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSpcAcceptCall()`, `TelSpcCallNumber()`

TelGetEvent Function

Purpose Retrieves events for applications that use the telephony manager.

Declared In `TelephonyMgr.h`

Prototype

```
void TelGetEvent(UInt16 iRefnum, TelAppID iAppId,
    EventPtr oEventP, Int32 iTimeOut)
```

Parameters

→ *`iRefnum`*
The telephony manager library reference number.

→ *`iAppId`*
The telephone application attachment identifier for your application.

← *`oEventP`*
A pointer to a `TelEventType` structure. Upon return, the structure contains the event information, which you should use as described in the Comments section.

→ *`iTimeout`*
Maximum number of ticks to wait before an event is returned (`evtWaitForever` means wait indefinitely).

Returns Returns nothing.

Comments This function retrieves both telephony and standard Palm OS® events. You must call this function to retrieve events in any application that is running in the UI task and using the telephony manager.

Upon return from this function, you need to test the type of the event by examining the `oEventP->type` field. If the event type is a telephony event, then you need to cast the pointer as follows to access the fields:

```
TelEventType *telEventP = (TelEventType *)oEventP;
```

This function calls both the `EvtGetEvent()` and `TelGetTelephonyEvent()` functions to retrieve the next event for your application.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `EvtGetEvent()`, `TelGetTelephonyEvent()`

TelGetTelephonyEvent Function

Purpose Retrieves telephony events only.

Declared In `TelephonyMgr.h`

Prototype

```
void TelGetTelephonyEvent(UInt16 iRefnum,
    TelAppID iAppId, EventPtr oEventP,
    Int32 iTimeOut)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

← *oEventP*
A pointer to a `TelEventType` structure. Upon return, the structure contains the event information.

→ *iTimeout*
Maximum number of ticks to wait before an event is returned (`evtWaitForever` means wait indefinitely).

Returns Returns nothing.

Comments Use this function instead of the `TelGetEvent()` function when you only want to process telephony events.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `EvtGetEvent()`, `TelGetEvent()`

TelInfGetInformation Function

Purpose Retrieve brand, model, and revision information for the phone.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelInfGetInformation(UInt16 iRefnum,
    TelAppID iAppId,
    TelInfGetInformationType *ioParamP,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioParamP*
A pointer to a `TelInfGetInformationType` structure.

On input, the `infoType` field of the structure contains the type of information that you want retrieved. The `size` field of this structure specifies the allocated size of the `value` buffer. Upon return, the `size` field specifies the actual size of the information that was retrieved, even if it was truncated to fit into the buffer.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` upon success or an error code upon failure.
`transId`	The transaction ID of the operation.
`paramP`	Points to the `TelInfGetInformationType` structure passed to this function in the `ioInfoP` parameter.
`functionId`	`kTelInfGetInformationMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments

Call this function to retrieve information about the currently connected phone.

The retrieved information is stored into the `value` field of the `TelInfGetInformationType` referenced by `ioInfoP` structure. If the `value` field buffer is too small to contain the complete information, the value is truncated and this function returns the `telErrBufferSize` error. The `size` field of the structure is always updated to contain the actual size of the retrieved information.

Before using this function, you should verify that it is available by calling the `TelIsInfServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

TelIsCfgServiceAvailable Macro

Purpose

Determines if the configuration service set is available in the current environment.

Declared In

`TelephonyMgr.h`

Prototype

```
#define TelIsCfgServiceAvailable (iRefnum,
    iAppId, ioTransIdP)
```

Parameters

→ *`iRefnum`*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the service set is available, or an error code if not.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` if the service set is available, or an error code if not.
`transId`	The transaction ID of the operation.
`paramP`	`kTelCfgServiceId`
`functionId`	`kTelUrqIsServiceAvailableMessage`

You need to call this macro before calling any function in the configuration service set, which is the family of functions that begin with the `TelCfg` prefix.

The configuration service set functions are documented in Chapter 77, "Telephony SMS."

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelIs<FunctionName>Supported()`

TelIsDtcServiceAvailable Macro

Purpose Determines if the data calls service set is available in the current environment.

Declared In `TelephonyMgr.h`

Prototype

```
#define TelIsDtcServiceAvailable (iRefnum,
    iAppId, ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the service set is available, or an error code if not.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` if the service set is available, or an error code if not.
`transId`	The transaction ID of the operation.
`paramP`	`kTelDtcServiceId`
`functionId`	`kTelUrqIsServiceAvailableMessage`

CommentsI You need to call this macro before calling any function in the data calls service set, which is the family of functions that begin with the `TelDtc` prefix.

The data calls service set functions are documented in Chapter 76, "Telephony Calls."

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelIs<FunctionName>Supported()`

TelIsEmcServiceAvailable Macro

Purpose Determines if the emergency calls service set is available in the current environment.

Declared In `TelephonyMgr.h`

Prototype

```
#define TelIsEmcServiceAvailable (iRefnum,
    iAppId, ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the service set is available, or an error code if not.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` if the service set is available, or an error code if not.
`transId`	The transaction ID of the operation.
`paramP`	`kTelEmcServiceId`
`functionId`	`kTelUrqIsServiceAvailableMessage`

Comments You need to call this macro before calling any function in the emergency calls service set, which is the family of functions that begin with the `TelEmc` prefix.

The emergency calls service set functions are documented in Chapter 76, "Telephony Calls."

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelIs<FunctionName>Supported()`

TelIsInfServiceAvailable Macro

Purpose Determines if the information service set is available in the current environment.

Declared In `TelephonyMgr.h`

Prototype

```
#define TelIsInfServiceAvailable (iRefnum,
    iAppId, ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the service set is available, or an error code if not.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` if the service set is available, or an error code if not.
`transId`	The transaction ID of the operation.

`paramP`	`kTelInfServiceId`
`functionId`	`kTelUrqIsServiceAvailableMessage`

Comments You need to call this macro before calling any function in the information service set, which is the family of functions that begin with the `TelInf` prefix.

The information service set functions are documented in this chapter.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelIs<FunctionName>Supported()`

TelIsNwkServiceAvailable Macro

Purpose Determines if the network service set is available in the current environment.

Declared In `TelephonyMgr.h`

Prototype

```
#define TelIsNwkServiceAvailable (iRefnum,
    iAppId, ioTransIdP)
```

Parameters

→ *`iRefnum`*
The telephony manager library reference number.

→ *`iAppId`*
The telephone application attachment identifier for your application.

↔ *`ioTransIdP`*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the service set is available, or an error code if not.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` if the service set is available, or an error code if not.
`transId`	The transaction ID of the operation.
`paramP`	`kTelNwkServiceId`
`functionId`	`kTelUrqIsServiceAvailableMessage`

Comments You need to call this macro before calling any function in the network service set, which is the family of functions that begin with the `TelNwk` prefix.

The network service set functions are documented in Chapter 75, "Telephony Network."

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelIs<FunctionName>Supported()`

TelIsOemServiceAvailable Macro

Purpose Determines if the OEM service set is available in the current environment.

Declared In `TelephonyMgr.h`

Prototype

```
#define TelIsOemServiceAvailable
    (iRefnum, iAppId, ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the service set is available, or an error code if not.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` if the service set is available, or an error code if not.
`transId`	The transaction ID of the operation.
`paramP`	`kTelOemServiceId`
`functionId`	`kTelUrqIsServiceAvailableMessage`

Comments You need to call this macro before calling any function in the OEM service set, which is the family of functions that begin with the `TelOem` prefix.

The OEM service set functions are documented in this chapter.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelIs<FunctionName>Supported()`

TelIsPhbServiceAvailable Macro

Purpose Determines if the phone book service set is available in the current environment.

Declared In `TelephonyMgr.h`

Prototype

```
#define TelIsPhbServiceAvailable
    (iRefnum, iAppId, ioTransIdP)
```

Parameters

→ *`iRefnum`*
The telephony manager library reference number.

→ *`iAppId`*
The telephone application attachment identifier for your application.

↔ *`ioTransIdP`*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points

to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the service set is available, or an error code if not.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` if the service set is available, or an error code if not.
`transId`	The transaction ID of the operation.
`paramP`	`kTelPhbServiceId`
`functionId`	`kTelUrqIsServiceAvailableMessage`

Comments

You need to call this macro before calling any function in the phone book service set, which is the family of functions that begin with the `TelPhb` prefix.

The phone book service set functions are documented in Chapter 78, "Telephony Phone Book."

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelIs<FunctionName>Supported()`

TelIsPhoneConnected Function

Purpose

Determines if a phone is connected.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelIsPhoneConnected(UInt16 iRefnum,
   TelAppID iAppId, UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` upon success or an error code upon failure.
`transId`	The transaction ID of the operation.
`paramP`	A `NULL` pointer.
`functionId`	`kTelUrqIsPhoneConnectedMessage`

Comments

Call this function to determine if there is currently a phone connected.

Compatibility

Implemented only if 4.0 New Feature Set is present.

TelIsPowServiceAvailable Macro

Purpose

Determines if the power services set is available in the current environment.

Declared In

`TelephonyMgr.h`

Prototype

```
#define TelIsPowServiceAvailable
    (iRefnum, iAppId, ioTransIdP)
```

Parameters

→ *iRefnum*

The telephony manager library reference number.

→ *iAppId*

The telephone application attachment identifier for your application.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the service set is available, or an error code if not.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` if the service set is available, or an error code if not.
`transId`	The transaction ID of the operation.
`paramP`	`kTelPowServiceId`
`functionId`	`kTelUrqIsServiceAvailableMessage`

Comments

You need to call this macro before calling any function in the power service set, which is the family of functions that begin with the `TelPow` prefix.

The power service set functions are documented in this chapter.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelIs<FunctionName>Supported()`

TelIsSmsServiceAvailable Macro

Purpose

Determines if the Short Message Service (SMS) service set is available in the current environment.

Declared In

`TelephonyMgr.h`

Prototype

```
#define TelIsSmsServiceAvailable
    (iRefnum, iAppId, ioTransIdP)
```

Parameters

→ *iRefnum*

The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the service set is available, or an error code if not.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` if the service set is available, or an error code if not.
`transId`	The transaction ID of the operation.
`paramP`	`kTelSmsServiceId`
`functionId`	`kTelUrqIsServiceAvailableMessage`

Comments

You need to call this macro before calling any function in the SMS service set, which is the family of functions that begin with the `TelSms` prefix.

The SMS service set functions are documented in Chapter 77, "Telephony SMS."

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelIs<FunctionName>Supported()`

TelIsSndServiceAvailable Macro

Purpose Determines if the sound service set is available in the current environment.

Declared In `TelephonyMgr.h`

Prototype

```
#define TelIsSndServiceAvailable
    (iRefnum, iAppId, ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the service set is available, or an error code if not.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` if the service set is available, or an error code if not.
`transId`	The transaction ID of the operation.
`paramP`	`kTelSndServiceId`
`functionId`	`kTelUrqIsServiceAvailableMessage`

Comments You need to call this macro before calling any function in the sound service set, which is the family of functions that begin with the `TelSnd` prefix.

The sound service set functions are documented in this chapter.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelIs<FunctionName>Supported()`

TelIsSpcServiceAvailable Macro

Purpose

Determines if the speech telephone call service set is available in the current environment.

Declared In

`TelephonyMgr.h`

Prototype

```
#define TelIsSpcServiceAvailable
    (iRefnum, iAppId, ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the service set is available, or an error code if not.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` if the service set is available, or an error code if not.
`transId`	The transaction ID of the operation.
`paramP`	`kTelSpcServiceId`
`functionId`	`kTelUrqIsServiceAvailableMessage`

Comments

You need to call this macro before calling any function in the speech telephone call service set, which is the family of functions that begin with the `TelSpc` prefix.

The speech telephone call service set functions are documented in Chapter 76, "Telephony Calls."

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelIs<FunctionName>Supported()`

TelIsStyServiceAvailable Macro

Purpose Determines if the security service set is available in the current environment.

Declared In `TelephonyMgr.h`

Prototype

```
#define TelIsStyServiceAvailable
    (iRefnum, iAppId, ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the service set is available, or an error code if not.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` if the service set is available, or an error code if not.
`transId`	The transaction ID of the operation.
`paramP`	`kTelStyServiceId`
`functionId`	`kTelUrqIsServiceAvailableMessage`

Comments You need to call this macro before calling any function in the security service set, which is the family of functions that begin with the `TelSty` prefix.

The security service set functions are documented in Chapter 75, "Telephony Network."

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelIs<FunctionName>Supported()`

TelMatchPhoneDriver Function

Purpose Determines if the currently selected driver matches the connected phone.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelMatchPhoneDriver(UInt16 iRefnum,
    TelAppID iAppId, UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` upon success or an error code upon failure
`transId`	The transaction ID of the operation.

`paramP`	A `NULL` pointer.
`functionId`	`kTelUrqMatchPhoneDriverMessage`

Compatibility Implemented only if 4.0 New Feature Set is present.

TelOemCall Function

Purpose Pass a call to an OEM function.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelOemCall(UInt16 iRefnum, TelAppID iAppId,
   TelOemCallType *ioParamP, UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioParamP*
A pointer to a `TelOemCallType` structure that contains information about the OEM function call.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` upon success or an error code upon failure.
`transID`	matches the output value of the `ioTransIdP` parameter

`functionId`	matches the function ID in the `TelOemCallType` structure passed to this function in the `ioParamP` parameter
`paramId`	points to the `TelOemCallType` structure passed to this function in the `ioParamP` parameter

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments Call this function to send a request to an OEM function. The calling function and the OEM function are responsible for coordinating the parameter block that is passed in the `TelOemCallType` structure.

Before using this function, you should verify that it is available by calling the `TelIsOemServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

TelOpen Function

Purpose Open the telephony manager API to initialize telephony services and launch the telephony task.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelOpen(UInt16 iRefnum, UInt32 iVersnum,
   TelAppID *oAppIdP)
```

Parameters

→ *`iRefnum`*
The telephony manager library reference number.

→ *`iVersnum`*
The version number of the shared library for which your application was developed. See "Version Constants" on page 1678.

← *`oAppIdP`*
A pointer to an application ID value. Upon return, this is the application ID that you supply as a parameter to the any other telephony functions that you call.

Returns Returns `errNone` if the function was successful or returns an error code if not successful. The following errors can occur:

- the telephony task could not be found (`telErrTTaskNotFound`)
- the telephony task could not be launched (`telErrTTaskNotRunning`)
- the phone driver could not be found
- the shared library version is not valid

Comments You can only call this function synchronously. You must call this function or `TelOpenProfile()` before calling any other telephony manager functions.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelClose()`

TelOpenPhoneConnection Function

Purpose Open communications with the connected phone.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelOpenPhoneConnection(UInt16 iRefnum,
   TelAppID iAppId, UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` upon success or an error code upon failure.
`transId`	The transaction ID of the operation.
`paramP`	A `NULL` pointer.
`functionId`	`kTelUrqOpenPhoneConnectionMessage`

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelClosePhoneConnection()`

TelOpenProfile Function

Purpose

Open the Telephony Manager API to initialize telephony services using a particular Connection Manager profile and launch the telephony task.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelOpenProfile (UInt16 iRefnum,
   UInt32 iVersnum, UInt32 profileId,
   TelAppID *oAppIdP)
```

Parameters

→ *`iRefnum`*

The telephony manager library reference number.

→ *`iVersnum`*

The version number of the shared library for which your application was developed. The version number of the shared library for which your application was developed. See "Version Constants" on page 1678.

→ *`profileId`*

The Connection Manager profile ID to use to initialize the Telephony Manager. If 0, use the one currently specified in the Phone Preferences panel.

You cannot specify a specific phone profile if the Telephony Manager has already been initialized. If you do, the phone profile you specify must match the one used to initialize the Telephony Manager.

← *oAppIdP*
A pointer to an application ID value. Upon return, this is the application ID that you supply as a parameter to the any other telephony functions that you call.

Returns `errNone` upon successful or one of the following values:

`telErrVersion`
The *iVersnum* parameter is invalid.

`telErrProfileConflict`
The Telephony Manager was already initialized with a phone profile that is different from that specified in *profileId*.

`telErrTooManyApps`
The Telephony Manager's application table is full.

Comments You can only call this function synchronously, and it must be called before you call any other Telephony Manager functions. `TelOpen` calls this function.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelClose()`

TelPowGetBatteryStatus Function

Purpose Retrieves the status of the phone's battery.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelPowGetBatteryStatus(UInt16 iRefnum,
   TelAppID iAppId, UInt8 *oStatusP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

← *oStatusP*
A pointer to an unsigned byte value. Upon return, this is the battery status value, which is one of the Battery Status Constants.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` upon success or an error code upon failure.
`transId`	The transaction ID of the operation.
`paramP`	Points to the unsigned integer value passed to this function in the `oStatusP` parameter.
`functionId`	`kTelPowGetBatteryStatusMessage`

WARNING! When using this function asynchronously, you must ensure that the value referenced by `oStatusP` remains in memory until the asynchronous call completes.

Comments

Before using this function, you should verify that it is available by calling the `TelIsPowServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelPowGetPowerLevel()`

TelPowGetPowerLevel Function

Purpose

Retrieve the current level of the phone battery, as a percentage value.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelPowGetPowerLevel(UInt16 iRefnum,
   TelAppID iAppId, UInt8 *oPowerP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

← *oPowerP*
A pointer to an unsigned byte value. Upon return, this is the battery percentage value.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` upon success or an error code upon failure.
`transId`	The transaction ID of the operation.
`paramP`	Points to the unsigned integer value passed to this function in the `oPowerP` parameter.
`functionId`	`kTelPowGetPowerLevelMessage`

WARNING! When using this function asynchronously, you must ensure that the value referenced by `oPowerP` remains in memory until the asynchronous call completes.

Comments The returned percentage value is an integer value between 0 and 100.

Before using this function, you should verify that it is available by calling the `TelIsPowServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelPowGetBatteryStatus()`

TelPowSetPhonePower Function

Purpose Turns the phone on or off.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelPowSetPhonePower(UInt16 iRefnum,
    TelAppID iAppId, Boolean iPowerOn)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

→ *iPowerOn*
Set to `true` to turn the phone on, and set to `false` to turn the phone off.

Returns Returns `errNone` if the function was successful and an error code if not.

Comments This function can only be called synchronously.

Before using this function, you should verify that it is available by calling the `TelIsPowServiceAvailable()` macro.

This function corresponds to the `kTelPowSetPhonePowerMessage` function ID value.

Compatibility Implemented only if 4.0 New Feature Set is present.

TelSendCommandString Function

Purpose Sends a command string to the phone or to the network.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSendCommandString(UInt16 iRefnum,
   TelAppID iAppId,
   TelSendCommandStringType *ioParamP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioParamP*
A pointer to a command string structure of type `TelSendCommandStringType`.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` upon success or an error code upon failure.
`transId`	The transaction ID of the operation.

`paramP`	Points to the `TelSendCommandStringType` structure passed to this function in the `ioParam` parameter.
`functionId`	Matches the function ID in the `TelSendCommandStringType` referenced by the `ioParamP` structure.

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Compatibility Implemented only if 4.0 New Feature Set is present.

TelSndMute Function

Purpose Mute or un-mute an active telephone call.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSndMute(UInt16 iRefnum, TelAppID iAppId,
   Boolean iMuteOn, UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

→ *iMuteOn*
Set to `true` to mute the telephone call, or set to `false` to unmute the telephone call.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` upon success or an error code upon failure.
`transId`	The transaction ID of the operation.
`paramP`	Points to the Boolean value passed to this function in the `iMuteOn` parameter.
`functionId`	`kTelSndMuteMessage`

Comments Before using this function, you should verify that it is available by calling the `TelIsSndServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSndPlayKeyTone()`

TelSndPlayKeyTone Function

Purpose Play a keytone sound on the phone.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSndPlayKeyTone(UInt16 iRefnum,
    TelAppID iAppId,
    TelSndPlayKeyToneType *iParamP,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

→ *iParamP*
A pointer to a `TelSndPlayKeyToneType` structure that describes the tone to play.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` upon success or an error code upon failure.
`transId`	The transaction ID of the operation.
`paramP`	Points to the `TelSndPlayKeyToneType` structure passed to this function in the `iKeyToneP` parameter.
`functionId`	`kTelSndPlayKeyToneMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `iParamP` remains in memory until the asynchronous call completes.

Comments

Before using this function, you should verify that it is available by calling the `TelIsSndServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelSndStopKeyTone()`

TelSndStopKeyTone Function

Purpose Stop the playing of a keytone sound on the phone.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSndStopKeyTone(UInt16 iRefnum,
   TelAppID iAppId, UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` upon success or an error code upon failure.
`transId`	The transaction ID of the operation.
`paramP`	A `NULL` pointer.
`functionId`	`kTelSndStopKeyToneMessage`

Comments Before using this function, you should verify that it is available by calling the `TelIsSndServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSndPlayKeyTone()`

TelUnblockNotifications Function

Purpose Requests that any pending notifications be sent.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelUnblockNotifications (UInt16 iRefnum)
```

Parameters → *iRefnum*
The telephony library reference number.

Returns Always returns `errNone`.

Comments You can only call this function synchronously.

As the Telephony Manager receives notifications from the phone driver, it queues them up in priority order and sends them as deferred notifications. Normally, there are no pending notifications in the queue. You should only need to call this function in an error situation if you want to ensure that any pending notification be sent.

See Also "Notification Identifiers" on page 1680

Feature Support Functions

This section describes the functions that you can call to determine if a specific feature or function is supported in the current operating environment.

TelIsFunctionSupported Function

Purpose Checks if a function is supported by the phone, driver, and network.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelIsFunctionSupported (UInt16 iRefnum,
   TelAppID iAppId, UInt16 functionId,
   UInt16 *ioTransIdP)
```

Parameters → *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

→ *functionId*
Identifier of the function to check. Specify one of the Message Identifier Constants.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the specified function is supported.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` if the specified function is supported, or an error code if not.
`transId`	The transaction ID of the operation.
`paramP`	The function ID of the function for which you are testing. For example, if you call `TelIsCfgGetSmsCenterSupported`, the value of this field is `kTelCfgGetSmsCenterMessage.` The function ID value for each function is described in the documentation for the function.
`functionId`	`kTelUrqIsFunctionSupportedMessage`

Comments

The `TelephonyMgr.h` header file also defines a series of macros that call this function, passing in the appropriate function identifier. These macros have the form `TelIs`*`FunctionName`*`Supported(`*`telDesc`*`)`.

TelIs<FunctionName>Supported Macros

Purpose Determines if the specified function is supported.

Declared In `TelephonyMgr.h`

Prototype

```
#define TelIs<FunctionName>Supported
    (iRefnum, iAppId, ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the specified function is supported.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` if the specified function is supported, or an error code if not.
`transId`	The transaction ID of the operation.
`paramP`	The function ID of the function for which you are testing. For example, if you call `TelIsCfgGetSmsCenterSupported`, the value of this field is `kTelCfgGetSmsCenterMessage.` The function ID value for each function is described in the documentation for the function.
`functionId`	`kTelUrqIsFunctionSupportedMessage`

Comments This is a family of synchronous macros that test if a specific function is available in the current environment.

To use the macro, substitute a function name for the <*FunctionName*> portion of the macro name. You can substitute any Telephony Manager function name; for a complete list of the Telephony Manager functions, see "Summary of Telephony Manager" on page 250 in *Palm OS Programmer's Companion*, vol. II, *Communications*.

For example, to determine if the `TelNwkGetSignalLevel()` function is available in the current environment, call the `TelIsNwkGetSignalLevelSupported` macro.

NOTE: A service set can be available without all of its functions being available. Thus, if the `TelIs<ServiceSet>Available()` macro returns `true` for a specific service set, you know that the service set is available, but you need to call `TelIs<FunctionName>Supported()` to determine if a specific function is available.

This macro corresponds to the `kTelUrqIsFunctionSupportedMessage` function ID value.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelIs<ServiceSet>Available()`

TelIsServiceAvailable Function

Purpose Checks if a service set (group of related functions) is supported by the phone, driver, and network.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelIsServiceAvailable (UInt16 iRefnum,
    TelAppID iAppId, UInt16 serviceId,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

→ *serviceId*
: Identifier of the service set to check. Specify one of the `TelServices` constants.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the service set is available.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` if the service set is available, or an error code if not.
`transId`	The transaction ID of the operation.
`paramP`	The service ID of the service set for which you are testing. For example, if you call `TelIsCfgServiceAvailable`, the value of this field is `kTelCfgServiceId`. The service ID is one of the `TelServices` constants.
`functionId`	`kTelUrqIsServiceAvailableMessage`

Comments

The `TelephonyMgr.h` header file also defines a series of macros that call this function, passing in the appropriate service set identifier. These macros have the form `TelIs`*`ServiceName`*`ServiceAvailable(`*`telDesc`*`)`.

TelIs<ServiceSet>Available Macros

Purpose Determines if the specified service set is available.

Declared In `TelephonyMgr.h`

Prototype

```
#define TelIs<ServiceSet>Available
    (iRefnum, iAppId, ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the service set is available.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`	`errNone` if the service set is available, or an error code if not.
`transId`	The transaction ID of the operation.
`paramP`	The service ID of the service set for which you are testing. For example, if you call `TelIsCfgServiceAvailable`, the value of this field is `kTelCfgServiceId`. The service ID is one of the `TelServices` constants.
`functionId`	`kTelUrqIsServiceAvailableMessage`

Comments This is a family of synchronous macros that test if a specific service set is available. You must call the appropriate set availability function before calling a function in the set.

NOTE: A service set can be available without all of its functions being available. You can use this macro to determine the availability of a specific service set, which you might use to determine the configuration of your applications' user interface. To test if a specific function is supported, use the `TelIs<FunctionName>Supported()` macro.

You can call these specific macros to determine if the service set is available:

- `TelIsCfgServiceAvailable()` to determine if the configuration service set is available.
- `TelIsDtcServiceAvailable()` to determine if the data calls service set is available.
- `TelIsEmcServiceAvailable()` to determine if the emergency calls service set is available.
- `TelIsInfServiceAvailable()` to determine if the information service set is available.
- `TelIsNwkServiceAvailable()` to determine if the network service set is available.
- `TelIsOemServiceAvailable()` to determine if the OEM service set is available.
- `TelIsPhbServiceAvailable()` to determine if the phone book service set is available.
- `TelIsPowServiceAvailable()` to determine if the power service set is available.
- `TelIsSmsServiceAvailable()` to determine if the SMS service set is available.
- `TelIsSndServiceAvailable()` to determine if the sound service set is available.
- `TelIsSpcServiceAvailable()` to determine if the speech calls service set is available.
- `TelIsStyServiceAvailable()` to determine if the security service set is available.

Each of these macros corresponds to the `kTelUrqIsServiceSupportedMessage` function ID value.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelIs<FunctionName>Supported()`

74

Telephony Security and Configuration

This chapter describes the telephony security and configuration service sets of the telephony API.

For more information about the telephony manager basic services and the different service sets, see Chapter 73, "Telephony Basic Services."

This chapter describes:

- Telephony Security and Configuration Data Structures
- Telephony Security and Configuration Constants
- Telephony Security and Configuration Functions

For more information about using the telephony manager, see Chapter 10, "Telephony Manager," in *Palm OS Programmer's Companion*, vol. II, *Communications*.

Telephony Security and Configuration Data Structures

This section describes the data structures used with the telephony security and configuration service sets portion of the telephony API.

TelCfgGetPhoneNumberType Struct

Purpose The `TelCfgGetPhoneNumber()` function uses a `TelCfgGetPhoneNumberType` structure to retrieve the connected phone dial number.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelCfgGetPhoneNumberType {
   UInt8 size;
   Char *value;
} TelCfgGetPhoneNumberType
```

Fields

↔ `size`

The size of the `value` buffer.

When the structure is used as an input parameter, this is the allocated size, in bytes, of the `value` buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the `value` buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

← `value`

A buffer into which the dial number is stored.

Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a `telErrBufferSize` error.

TelCfgGetSmsCenterType Struct

Purpose The `TelCfgGetSmsCenter()` function uses a `TelCfgGetSmsCenterType` structure to retrieve the SMS service center dial number.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelCfgGetSmsCenterType {
   UInt8 size;
   Char *value;
} TelCfgGetSmsCenterType
```

Fields

↔ `size`

The size of the `value` buffer.

When the structure is used as an input parameter, this is the allocated size, in bytes, of the `value` buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the `value` buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

← `value`

A buffer into which the dial number is stored.

Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a `telErrBufferSize` error.

TelStyChangeAuthenticationType Struct

Purpose

You use the `TelStyChangeAuthenticationType` to change an authentication code with the `TelStyChangeAuthenticationCode()` function.

Declared In

`TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelStyChangeAuthenticationType {
   UInt8 codeId;
   Char *oldCode;
   Char *newCode;
} TelStyChangeAuthenticationType
```

Fields

→ `codeId`

The ID of the authentication code to change.

→ `oldCode`

The previous value of the code.

→ `newCode`

The new value of the code.

Telephony Security and Configuration Constants

This section describes the constants used with the telephony security and configuration service sets of the telephony API.

Authentication State Constants

Purpose The authentication state constants describe the current authentication state of the mobile unit connection.

Declared In `TelephonyMgr.h`

Constants

`#define kTelStyReady 0`
No additional security information is expected.

`#define kTelStyPin1CodeId 1`
The PIN1 code is expected.

`#define kTelStyPin2CodeId 2`
The PIN2 code is expected.

`#define kTelStyPuk1CodeId 3`
The PUK1 code is expected.

`#define kTelStyPuk2CodeId 4`
The PUK2 code is expected.

`#define kTelStyPhoneToSimCodeId 5`
The phone-to-SIM code is expected.

`#define kTelStyFirstOemCodeId 6`
An OEM code is expected.

The constant `kTelStyFirstOemCodeId` specifies the first OEM authentication code. You can specify additional OEM codes by incrementing this value. For example, to specify the third OEM authentication code, use `kTelStyFirstOemCodeId+2`.

Telephony Security and Configuration Functions

This section describes the data structures used with the telephony security and configuration service sets portion of the telephony API.

TelCfgGetPhoneNumber Function

Purpose

Retrieve the connected telephone number.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelCfgGetPhoneNumber(UInt16 iRefnum,
   TelAppID iAppId,
   TelCfgGetPhoneNumberType* ioParamP,
   UInt16* ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

↔ *ioParamP*
: A pointer to a `TelCfgGetPhoneNumberType` structure that is filled in with the dial telephone number.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelCfgGetPhoneNumberType` structure passed to this function in the `ioParamP` parameter.

`functionId`
: `kTelCfgGetPhoneNumberMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments The connected dial telephone number is stored into the `value` field of the `TelCfgGetPhoneNumberType` structure referenced by `ioParamP`. If the `value` field buffer is too small to contain the complete telephone number, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `size` field of the structure is always updated to contain the actual size of the complete telephone number.

Before using this function, you should verify that it is available by calling the `TelIsCfgServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelCfgSetSmsCenter()`, `TelSmsSendMessage()`

TelCfgGetSmsCenter Function

Purpose Retrieve the SMS service center dial telephone number.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelCfgGetSmsCenter(UInt16 iRefnum,
    TelAppID iAppId,
    TelCfgGetSmsCenterType* ioParamP,
    UInt16* ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioParamP*
A pointer to a `TelCfgGetSmsCenterType` structure that is filled in with the dial telephone number.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelCfgGetSmsCenterType` structure passed to this function in the `ioParamP` parameter.

`functionId`
: `kTelCfgGetSmsCenterMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments

The service center dial telephone number is stored into the `value` field of the `TelCfgGetSmsCenterType` structure referenced by `ioParamP`. If the `value` field buffer is too small to contain the complete telephone number, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `size` field of the structure is always updated to contain the actual size of the complete telephone number.

Before using this function, you should verify that it is available by calling the `TelIsCfgServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelCfgSetSmsCenter()`, `TelSmsSendMessage()`

TelCfgSetSmsCenter Function

Purpose Set the SMS service center dial telephone number.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelCfgSetSmsCenter(UInt16 iRefnum,
    TelAppID iAppId, const Char* iDialNumberP,
    UInt16* ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

→ *iDialNumberP*
A pointer to the null-terminated dial telephone number string for the SMS service center.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
`errNone` upon success or an error code upon failure.

`transId`
The transaction ID of the operation.

`paramP`
: Points to the string passed to this function in the `iDialNumberP` parameter.

`functionId`
: `kTelCfgSetSmsCenterMessage`

WARNING! When using this function asynchronously, you must ensure that the value referenced by `iDialNumberP` remains in memory until the asynchronous call completes.

Comments Before using this function, you should verify that it is available by calling the `TelIsCfgServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelCfgGetSmsCenter()`, `TelSmsSendMessage()`

TelStyChangeAuthenticationCode Function

Purpose Change the value of an authentication code. Note that you can only use this function with GSM networks.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelStyChangeAuthenticationCode(UInt16 iRefnum,
    TelAppID iAppId,
    TelStyChangeAuthenticationType* iParamP,
    UInt16* ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

→ *iParamP*
: A pointer to a `TelStyChangeAuthenticationType` structure that contains the old and new authentication code values.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelStyChangeAuthenticationType` structure passed to this function in the `iParamP` parameter.

`functionId`
: `kTelStyChangeAuthenticationCodeMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `iParamP` remains in memory until the asynchronous call completes.

Comments

Before using this function, you should verify that it is available by calling the `TelIsStyServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelStyEnterAuthenticationCode()`

TelStyEnterAuthenticationCode Function

Purpose

Enter the authentication code for which the phone is currently waiting. Note that you can only use this function with GSM networks.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelStyEnterAuthenticationCode(UInt16 iRefnum,
    TelAppID iAppId, const Char* iCodeP,
    UInt16* ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

→ *iCodeP*
: A pointer to the null-terminated authentication code string to send to the phone.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the string passed to this function in the `iCodeP` parameter.

```
functionId
     kTelStyEnterAuthenticationCodeMessage
```

WARNING! When using this function asynchronously, you must ensure that the string referenced by `iCodeP` remains in memory until the asynchronous call completes.

Comments Before using this function, you should verify that it is available by calling the `TelIsStyServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelStyChangeAuthenticationCode()`, `TelStyGetAuthenticationState()`

TelStyGetAuthenticationState Function

Purpose Returns the current state of user authentication.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelStyGetAuthenticationState(UInt16 iRefnum,
   TelAppID iAppId, UInt8* oStateP,
   UInt16* ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

← *oStateP*
A pointer to an unsigned byte value. Upon return, this is the authentication state, which is one of the Authentication State Constants.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `oStateP` parameter.

`functionId`
: `kTelStyGetAuthenticationStateMessage`

WARNING! When using this function asynchronously, you must ensure that the value referenced by `oStateP` remains in memory until the asynchronous call completes.

Comments

Before using this function, you should verify that it is available by calling the `TelIsStyServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelStyEnterAuthenticationCode()`

75

Telephony Network

This chapter describes the telephony network service set of the telephony API.

For more information about the telephony manager basic services and the different service sets, see Chapter 73, "Telephony Basic Services."

This chapter describes:

- Telephony Network Data Structures
- Telephony Network Constants
- Telephony Network Functions

For more information about using the telephony manager, see Chapter 10, "Telephony Manager," in *Palm OS Programmer's Companion*, vol. II, *Communications*.

Telephony Network Data Structures

This section describes the data structures used with the telephony network service set portion of the telephony API.

TelNwkGetLocationType Struct

Purpose You use the `TelNwkGetLocationType` structure with the `TelNwkGetLocation()` function to retrieve information about the location of the phone.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelNwkGetLocationType {
   Char *value;
   UInt16 size;
} TelNwkGetLocationType
```

Fields ← `value`

A buffer into which the current location string is stored. The format of this string is network-dependent.

Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a `telErrBufferSize` error.

↔ `size`

The size of the `value` string. When the structure is used as an input parameter, this is the allocated size, in bytes, of the `value` buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the `value` buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

TelNwkGetNetworkNameType Struct

Purpose You use the `TelNwkGetNetworkNameType` structure with the `TelNwkGetNetworkName()` function to retrieve the name of the specified, registered network.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelNwkGetNetworkNameType {
   UInt32 id;
   Char *value;
   UInt16 size;
} TelNwkGetNetworkNameType
```

Fields → `id`

The network ID.

← `value`

A null-terminated string buffer into which the network name is stored.

Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a `telErrBufferSize` error.

↔ `size`

The size of the `value` string. When the structure is used as an input parameter, this is the allocated length, in bytes, of the `value` buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the `value` buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error

TelNwkGetNetworksType Struct

Purpose You use the `TelNwkGetNetworksType` structure with the `TelNwkGetNetworks()` function to retrieve the number of registered networks.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelNwkGetNetworksType {
   UInt32 *networkIdP;
   UInt8 size;
} TelNwkGetNetworksType
```

Fields

← `networkIdP`

An array into which the retrieved network IDs are stored.

Note that if this buffer is too small to contain all of the available IDs, the data is truncated and the function using this structure generates a `telErrBufferSize` error.

↔ `size`

When the structure is use as an input value, this is the allocated size, in elements, of the `networkIdP` array.

When the structure is used as a return value, this is the number of network IDs that are available. If the `networkP` buffer is too small to contain all of the retrieved IDs, this field is assigned the entire number of available IDs, and the function using this structure generates a `telErrBufferSize` error

Telephony Network Constants

This section describes the constants used with the telephony network service set of the telephony API, which include the following constant types:

- Network Type Constants
- Network Search Mode Constants

Network Type Constants

Purpose The network type constants describe the type of network connection.

Declared In `TelephonyMgr.h`

Constants

`#define kTelNwkCDMA 0`
A CDMA network.

`#define kTelNwkGSM 1`
A GSM network.

`#define kTelNwkTDMA 2`
A TDMA network.

`#define kTelNwkPDC 3`
A PDC network.

Network Search Mode Constants

Purpose The network search mode constants describe the search mode used to find a network.

Declared In `TelephonyMgr.h`

Constants

`#define kTelNwkAutomaticSearch 0`
Automatic network searching.

`#define kTelNwkManualSearch 1`
Manual network searching.

Telephony Network Functions

This section describes the data structures used with the telephony network service set portion of the telephony API.

TelNwkGetLocation Function

Purpose Retrieve information about the location of the mobile unit.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelNwkGetLocation(UInt16 iRefnum,
   TelAppID iAppId,
   TelNwkGetLocationType* ioParamP,
   UInt16* ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioParamP*
A pointer to a `TelNwkGetLocationType` structure.

On input, the `size` field of this structure specifies the allocated size of the `value` buffer. Upon return, the `size` field specifies the actual size of the location string, even if it was truncated to fit into the buffer.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelNwkGetLocationType` passed to this function in the `ioParamP` parameter.

`functionId`
: `kTelNwkGetLocationMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments The location information string is stored into the `value` field of the `TelNwkGetLocationType` structure referenced by `ioParamP`. If the `value` buffer is too small to contain the complete string, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `size` field of the structure is always updated to contain the actual size of the complete string.

Before using this function, you should verify that it is available by calling the `TelIsNwkServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

TelNwkGetNetworkName Function

Purpose Returns the name of a registered network.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelNwkGetNetworkName(UInt16 iRefnum,
    TelAppID iAppId,
    TelNwkGetNetworkNameType* ioParamP,
    UInt16* ioTransIdP)
```

Parameters

→ *`iRefnum`*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

↔ *ioParamP*
: A pointer to a `TelNwkGetNetworkNameType` structure that stores the network name.

  On input, the `size` field of this structure specifies the allocated size of the `value` buffer. Upon return, the `size` field specifies the actual size of the name string, even if it was truncated to fit into the buffer.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelNwkGetNetworkNameType` passed to this function in the `ioParamP` parameter.

`functionId`
: `kTelNwkGetNetworkNameMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments

The network name string is stored into the `value` field of the structure. If the `value` field buffer in the

TelNwkGetNetworkNameType structure is too small to contain the complete string, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `size` field is always updated to contain the actual size of the complete string.

The string that is returned in the `value` field of the structure referenced by `ioParamP` is network dependent.

On a GSM network, the result string is compliant with the AT 07.07 European Telecommunications Standards Institute (ETSI) standard for `COPS` and `CREG` commands. The result string contains the following elements:

- The network type, as returned by the TelNwkGetNetworkType() function, and followed by a semicolon (`';'`) character.
- The network operator, using the following syntax:

```
<area code> ';' <network registration>
```

The `<area code>` value is the value retrieved by issuing the `AT+CREG?` command.

The `<network registration>` value is the value retrieved by issuing the `AT+CREG?` command.

Before using this function, you should verify that it is available by calling the TelIsNwkServiceAvailable() macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also TelNwkGetNetworks(), TelNwkGetSelectedNetwork()

TelNwkGetNetworks Function

Purpose Retrieves information about the registered networks.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelNwkGetNetworks(UInt16 iRefnum,
   TelAppID iAppId,
   TelNwkGetNetworksType* ioParamP,
   UInt16* ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioParamP*
A pointer to a `TelNwkGetNetworksType` structure that stores the network IDs. On input, the `size` field of this structure contains the size, in elements, of the `networkIdP` array field.

Upon return, the `networkIdP` array contains the IDs of the registered networks, and the `size` field contains the number of IDs in the array.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
`errNone` upon success or an error code upon failure.

`transId`
The transaction ID of the operation.

`paramP`
: Points to the `TelNwkGetNetworksType` passed to this function in the `ioParamP` parameter.

`functionId`
: `kTelNwkGetNetworksMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments Before using this function, you should verify that it is available by calling the `TelIsNwkServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelNwkGetNetworkName()`, `TelNwkGetNetworks()`

TelNwkGetNetworkType Function

Purpose Retrieve the type of the selected network.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelNwkGetNetworkType(UInt16 iRefnum,
    TelAppID iAppId, UInt8* oTypeP,
    UInt16* ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

← *oTypeP*
: A pointer to an unsigned byte value. Upon return, this is the network type. This is one of Network Type Constants.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points

to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `oTypeP` parameter.

`functionId`
: `kTelNwkGetNetworkTypeMessage`

WARNING! When using this function asynchronously, you must ensure that the value you pass for the `oTypeP` parameter remains in memory until the asynchronous call completes.

Comments

Before using this function, you should verify that it is available by calling the `TelIsNwkServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

TelNwkGetSearchMode Function

Purpose

Returns the current network search mode.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelNwkGetSearchMode(UInt16 iRefnum,
    TelAppID iAppId, UInt8* oModeP,
    UInt16* ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

← *oModeP*
: A pointer to an unsigned byte value. Upon return, this is the type of search mode that is currently being used. This is one of the Network Search Mode Constants.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `oModeP` parameter.

`functionId`
: `kTelNwkGetSearchModeMessage`

WARNING! When using this function asynchronously, you must ensure that the value referenced by `oModeP` remains in memory until the asynchronous call completes.

Comments

Before using this function, you should verify that it is available by calling the `TelIsNwkServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelNwkSetSearchMode()`

TelNwkGetSelectedNetwork Function

Purpose Retrieve the network identifier of the currently selected network.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelNwkGetSelectedNetwork(UInt16 iRefnum,
   TelAppID iAppId, UInt32* oNetworkIdP,
   UInt16* ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

← *oNetworkIdP*
: A pointer to an unsigned integer value. Upon return, this is the identifier of the currently selected network.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `oNetworkIdP` parameter.

`functionId`
: `kTelNwkGetSelectedNetworkMessage`

WARNING! When using this function asynchronously, you must ensure that the value referenced by `oNetworkP` remains in memory until the asynchronous call completes.

Comments Before using this function, you should verify that it is available by calling the `TelIsNwkServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelNwkGetNetworkName()`, `TelNwkGetNetworks()`, `TelNwkSelectNetwork()`

TelNwkGetSignalLevel Function

Purpose Retrieve the selected network carrier signal level.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelNwkGetSignalLevel(UInt16 iRefnum,
   TelAppID iAppId, UInt8* oSignalP,
   UInt16* ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

← *oSignalP*
A pointer to an unsigned byte value. Upon return, this is an indication of the signal level in decibels per milliwatt (dBm). The values are explained in the Comments section.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `oSignalP` parameter.

`functionId`
: `kTelNwkGetSignalLevelMessage`

WARNING! When using this function asynchronously, you must ensure that the value referenced by `oSignalP` remains in memory until the asynchronous call completes.

Comments

This function sets the value of the variable referenced by `oSignalP` to an integer value that indicates the signal strength in dBm.

The following table describes the signal strength values.

Table 75.1 Network carrier singal strength values

Signal level value	dBm value
0	≤ –113 dBm
1	–111 dBm
2 to 30	–109 dBm to –53 dBm
31	≥ –51 dBm
99	unknown or undetectable

Before using this function, you should verify that it is available by calling the `TelIsNwkServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

TelNwkSelectNetwork Function

Purpose Select a network to use from among the set of registered networks.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelNwkSelectNetwork(UInt16 iRefnum,
   TelAppID iAppId, UInt32 iNetworkId,
   UInt16* ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

→ *iNetworkId*
The identifier of the network to select.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
`errNone` upon success or an error code upon failure.

`transId`
The transaction ID of the operation.

`paramP`
Points to the network unsigned integer value passed to this function in the `iNetworkId` parameter.

`functionId`
`kTelNwkSelectNetworkMessage`

Comments Before using this function, you should verify that it is available by calling the `TelIsNwkServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelNwkGetNetworkName()`, `TelNwkGetNetworks()`, `TelNwkGetSelectedNetwork()`

TelNwkSetSearchMode Function

Purpose Sets the search mode used to find a network.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelNwkSetSearchMode(UInt16 iRefnum,
   TelAppID iAppId, UInt8 iMode,
   UInt16* ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

→ *iMode*
The search mode to use. This must be one of the Network Search Mode Constants.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
`errNone` upon success or an error code upon failure.

`transId`
The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `iMode` parameter.

`functionId`
: `kTelNwkSetSearchModeMessage`

Comments Before using this function, you should verify that it is available by calling the `TelIsNwkServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelNwkGetSearchMode()`

76

Telephony Calls

This chapter describes the telephony calls service sets of the telephony API, which include:

- The data calls service set
- The emergency calls service set
- The speech (voice) calls service set

For more information about the telephony manager basic services and the different service sets, see Chapter 73, "Telephony Basic Services."

This chapter describes:

- Telephony Calls Data Structures
- Telephony Calls Functions

For more information about using the telephony manager, see Chapter 10, "Telephony Manager," in *Palm OS Programmer's Companion*, vol. II, *Communications*.

Telephony Calls Data Structures

This section describes the data structures used with the telephony calls service sets portion of the telephony manager API.

TelDtcCallNumberType Struct

Purpose The `TelDtcCallNumber()` function uses the `TelDataCallNumberType` structure to specify information about the telephone number to call.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelDtcCallNumberType {
   Char *dialNumberP;
   UInt8 lineId;
} TelDtcCallNumberType
```

Fields

→ `dialNumberP`
The telephone number to dial.

← `lineId`
Upon return from the `TelDtcCallNumber()` function, this is the ID of the data line that was established for the telephone call.

TelDtcReceiveDataType Struct

Purpose The `TelDtcReceiveData()` function uses a `TelDtcReceiveDataType` structure to receive data from an open data line.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelDtcReceiveDataType {
   UInt8 *data;
   UInt32 size;
   UInt32 timeOut;
} TelDtcReceiveDataType
```

Fields

← `data`
A buffer into which the data is stored.

Note that if this buffer is too small to contain all of the available data, the end of the data is truncated and the function using this structure generates a `telErrBufferSize` error.

↔ `size`
: When the structure is used as an input parameter, this is the allocated size, in bytes, of the `data` buffer.

  Upon return, this is the actual number of bytes of data that was retrieved. If the `data` buffer is too small to contain all of the retrieved data, the function using this structure generates a `telErrBufferSize` error.

→ `timeout`
: The number of milliseconds to wait before timing out.

TelDtcSendDataType Struct

Purpose The `TelDtcReceiveData()` function uses a `TelDtcSendDataType` structure to send data to an open data line.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelDtcSendDataType {
   UInt8 *data;
   UInt32 size;
} TelDtcSendDataType
```

Fields

→ `data`
: A pointer to the data to send.

→ `size`
: The number of bytes of `data` in the data buffer.

TelEmcGetNumberType Struct

Purpose The `TelEmcGetNumber()` function uses a `TelEmcGetNumberType` structure to retrieve an emergency dial telephone number.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelEmcGetNumberType {
   UInt8 index;
   UInt8 size;
   Char *value;
} TelEmcGetNumberType
```

Fields

→ `index`
The index of the telephone number. This is a zero-based index.

↔ `size`
When the structure is used as an input parameter, this is the allocated size, in bytes, of the `value` buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the `value` buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

← `value`
A null-terminated string buffer into which the emergency dial telephone number is stored.

Note that if this buffer is too small to contain all of the available data, the end of the data is truncated and the function using this structure generates a `telErrBufferSize` error.

TelEmcSetNumberType Struct

Purpose

The `TelEmcSetNumber()` function uses a `TelEmcNumberType` structure to set an emergency dial telephone number.

Declared In

`TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelEmcSetNumberType {
   UInt8 index;
   Char *value;
} TelEmcSetNumberType
```

Fields

→ `index`
The index of the telephone number. This is a zero-based index.

→ `value`
The string value of the number to store as the `index`th entry.

TelSpcGetCallerNumberType Struct

Purpose The `TelSpcGetCallerNumber()` function uses a `TelSpcGetCallerNumberType` structure to retrieve an incoming telephone number.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSpcGetCallerNumberType {
   UInt8 size;
   Char *value;
} TelSpcGetCallerNumberType
```

Fields

↔ `size`

When the structure is used as an input parameter, this is the allocated size, in bytes, of the `value` buffer.

Upon return, this is the actual size of the caller dial telephone number, including the null terminator character. If the `value` buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

← `value`

A null-terminated string buffer into which the caller telephone number is stored.

Note that if this buffer is too small to contain all of the available data, the end of the data is truncated and the function using this structure generates a `telErrBufferSize` error.

TelSpcPlayDTMFType Struct

Purpose The `TelSpcPlayDTMF()` function uses a `TelSpcPlayDTMFType` structure to specify the qualities of the DTMF (dual-tone, multi-frequency) sound sent by the phone to the network or remote, connected equipment.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSpcPlayDTMFType {
   UInt8 keyTone;
   UInt32 duration;
} TelSpcPlayDTMFType
```

Fields

→ `keyTone`
The keycode of the key tone to play. This must be one of the values declared under "Keycode Constants" on page 1669.

→ `duration`
The duration of the tone, specified as a multiple of ten milliseconds.

Telephony Calls Functions

This section describes the functions used with the telephony calls service sets portion of the telephony API.

TelDtcCallNumber Function

Purpose
Initiate a data telephone call.

Declared In
`TelephonyMgr.h`

Prototype

```
Err TelDtcCallNumber(UInt16 iRefnum,
   TelAppID iAppId,TelDtcCallNumberType *ioParamP,
   UInt16 *ioTransIdP)
```

Parameters

→ *`iRefnum`*
The telephony manager library reference number.

→ *`iAppId`*
The telephone application attachment identifier for your application.

↔ *`ioParamP`*
A pointer to a `TelDtcCallNumberType` structure that specifies information about the telephone call.

↔ *`ioTransIdP`*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result
Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelDtcCallNumberType` structure passed to this function in the `ioDataCallParamP` parameter.

`functionId`
: `kTelDtcCallNumberMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioDataCallParamP` remains in memory until the asynchronous call completes.

Comments

Call this function to start a data telephone call.

Before using this function, you should verify that it is available by calling the `TelIsDtcServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelDtcCloseLine()`

TelDtcCloseLine Function

Purpose

Hang up a data line.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelDtcCloseLine(UInt16 iRefnum,
   TelAppID iAppId, UInt8 iLineId,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

→ *iLineId*
: The ID of the line to hang up. This is the ID returned by a previous call to the `TelDtcCallNumber()` function.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `iLineId` parameter.

`functionId`
: `kTelDtcCloseLineMessage`

Call this function to end a data telephone call.

Before using this function, you should verify that it is available by calling the `TelIsDtcServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelDtcCallNumber()`

TelDtcReceiveData Function

Purpose Receive data on an opened data communications line.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelDtcReceiveData(UInt16 iRefnum,
   TelAppID iAppId,
   TelDtcReceiveDataType *ioParamP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioParamP*
A pointer to a `TelDtcReceiveDataType` structure.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
`errNone` upon success or an error code upon failure.

`transId`
The transaction ID of the operation.

`paramP`
Points to the `TelDtcReceiveDataType` structure passed to this function in the `ioRcvDataP` parameter.

`functionId`
`kTelDtcReceiveDataMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments Call this function to receive data during an active data telephone call.

Before using this function, you should verify that it is available by calling the `TelIsDtcServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelDtcCallNumber()`, `TelDtcCloseLine()`, `TelDtcSendData()`

TelDtcSendData Function

Purpose Send data on an opened data line.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelDtcSendData(UInt16 iRefnum, TelAppID iAppId,
    TelDtcSendDataType *iParamP,UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

→ *iParamP*
A pointer to a `TelDtcSendDataType` structure that specifies the data to send.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelDtcSendDataType` structure passed to this function in the `iParamP` parameter.

`functionId`
: `kTelDtcSendDataMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `iParamP` remains in memory until the asynchronous call completes.

Comments

Call this function to send data during an active data telephone call.

Before using this function, you should verify that it is available by calling the `TelIsDtcServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelDtcCallNumber()`, `TelDtcCloseLine()`, `TelDtcReceiveData()`

TelEmcCall Function

Purpose

Call the currently selected emergency service.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelEmcCall(UInt16 iRefnum, TelAppID iAppId,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: A `NULL` pointer.

`functionId`
: `kTelEmcCallMessage`

Comments

This function calls the telephone number specified in a previous call to the `TelEmcSelectNumber()` function. In synchronous mode, this function returns after the dial command has been sent to the phone.

After calling this function, sub-launched applications can receive notifications when the following telephony events occur. Note that these notifications can be raised after both synchronous and asynchronous calls to this function.

Table 76.1 Telephony events sent to sublaunched applications

Event	Description
`sysTelSpcLaunchCmdCallReleased`	Warns that the telephone call has been released.
`sysTelSpcLaunchCmdCallBusy`	Warns that the called equipment is busy.

Table 76.1 Telephony events sent to sublaunched applications

Event	Description
`sysTelSpcLaunchCmdCallConnect`	Warns that the line is open. The ID of the open line is stored in the `UInt32` value of the parameter block passed to the application.
`sysTelSpcLaunchCmdCallError`	Warns that an error occurred while attempting to complete the telephone call.

Before using this function, you should verify that it is available by calling the `TelIsEmcServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelEmcCloseLine()`,

TelEmcCloseLine Function

Purpose Close the line that is currently opened for an emergency telephone call.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelEmcCloseLine(UInt16 iRefnum,
   TelAppID iAppId, UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: A `NULL` pointer.

`functionId`
: `kTelDtcCloseLineMessage`

Comments

Call this function to end an emergency telephone call.

Before using this function, you should verify that it is available by calling the `TelIsEmcServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelEmcCall()`

TelEmcGetNumber Function

Purpose

Retrieve an emergency dial telephone number.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelEmcGetNumber(UInt16 iRefnum,
    TelAppID iAppId, TelEmcGetNumberType *ioParamP,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

↔ *ioParamP*
: A pointer to a `TelEmcGetNumberType` structure in which you assign the index of the telephone number that you want to retrieve.

  On input, the `size` field of this structure specifies the allocated size of the `value` buffer. Upon return, the `size`

field specifies the actual size of the telephone number, even if it was truncated to fit into the buffer.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelEmcGetNumberType` structure passed to this function in the `ioGetNumberP` parameter.

`functionId`
: `kTelEmcGetNumberMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments

The emergency call telephone number is stored into the `value` field of the `TelEmcGetNumberType` structure referenced by `ioGetNumberP`. If the `value` buffer is too small to contain the complete string, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `size` field of the structure is always updated to contain the actual size of the complete string.

Before using this function, you should verify that it is available by calling the `TelIsEmcServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelEmcGetNumberCount()`, `TelEmcSetNumber()`, `TelEmcSelectNumber()`

TelEmcGetNumberCount Function

Purpose Retrieve the count of emergency dial telephone numbers.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelEmcGetNumberCount(UInt16 iRefnum,
   TelAppID iAppId, UInt8 *oCountP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

← *oCountP*
Upon return, the total number of emergency call numbers available.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
`errNone` upon success or an error code upon failure.

`transId`
The transaction ID of the operation.

`paramP`
: Points to unsigned integer passed to this function in the `oCountP` parameter.

`functionId`
: `kTelEmcGetNumberMessage`

WARNING! When using this function asynchronously, you must ensure that the value referenced by `oCountP` remains in memory until the asynchronous call completes.

Comments

The emergency telephone call number is stored into the `value` field of the `TelEmcGetNumberType` structure referenced by `ioGetNumberP`. If the `value` buffer is too small to contain the complete string, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `size` field of the structure is always updated to contain the actual size of the complete string.

Before using this function, you should verify that it is available by calling the `TelIsEmcServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelEmcGetNumber()`, `TelEmcSetNumber()`, `TelEmcSelectNumber()`

TelEmcSelectNumber Function

Purpose

Select the current emergency telephone number. This is the telephone number that gets dialed when you call the `TelEmcCall()` function.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelEmcSelectNumber(UInt16 iRefnum,
   TelAppID iAppId, UInt8 iIndex,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

→ *iIndex*
: The zero-based index of the emergency telephone number that you want selected.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `iIndex` parameter.

`functionId`
: `kTelEmcSelectNumberMessage`

Comments

Before using this function, you should verify that it is available by calling the `TelIsEmcServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelEmcCall()`, `TelEmcGetNumber()`, `TelEmcGetNumberCount()`, `TelEmcSetNumber()`

TelEmcSetNumber Function

Purpose Set the telephone number for the specified emergency dial number.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelEmcSetNumber(UInt16 iRefnum,
   TelAppID iAppId, TelEmcSetNumberType *iParamP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

→ *iParamP*
: A pointer to a `TelEmcSetNumberType` structure that specifies the telephone number.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelEmcSetNumberType` structure passed to this function in the `iNumberP` parameter.

`functionId`
: `kTelEmcSetNumberMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `iParamP` remains in memory until the asynchronous call completes.

Comments Call this function to associate a new telephone number with the emergency dial number that has the specified `iIndex`.

Before using this function, you should verify that it is available by calling the `TelIsEmcServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelEmcGetNumber()`,

TelSpcAcceptCall Function

Purpose Accept an incoming voice telephone call.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSpcAcceptCall(UInt16 iRefnum,
    TelAppID iAppId, UInt8 *oLineIdP,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

← *oLineIdP*
A pointer to an unsigned byte value. Upon return, this is the ID of the voice line assigned to the telephone call.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer passed to this function in the `oLineIdP` parameter.

`functionId`
: `kTelSpcAcceptCallMessage`

WARNING! When using this function asynchronously, you must ensure that the value referenced by `oLineIdP` remains in memory until the asynchronous call completes.

Comments

If another line was active prior to the execution of this function, that line is put on hold. Note that there can only be one line active at any given time, and there can only be one line on hold at any given time.

Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelSpcGetCallerNumber()`, `TelSpcRejectCall()`

TelSpcCallNumber Function

Purpose

Initiate a voice telephone call.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelSpcCallNumber(UInt16 iRefnum,
    TelAppID iAppId, const Char *iDialNumberP,
    UInt16 *ioTransIdP)
```

Parameters

→ *`iRefnum`*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

→ *iDialNumberP*
: A pointer to the telephone number to call.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the string passed to this function in the `iDialNumberP` parameter.

`functionId`
: `kTelSpcCallNumberMessage`

WARNING! When using this function asynchronously, you must ensure that the value referenced by `iDialNumberP` remains in memory until the asynchronous call completes.

Comments

A successful return from a synchronous call or receipt of a successful notification from an asynchronous call does not mean that the telephone call has been connected; instead, it indicates that the dial command was sent to the phone. Successful connection of the telephone call is indicated with a sub-launch.

The dial number is formatted according to the following syntax:

```
DialNumber ::=  <Phone_Number> | <Code_String>
               | <Phone_Number> <Code_String>

Phone_Number ::=  <IntlPrefix><NatlNumber>
                | <NatlNumber>

IntlPrefix ::=  '+' <country code>

NatlNumber ::=  {{Pause<Pause>}{<Digit>}}

Code_String ::=  <Symbol>{<Symbol>}

Symbol ::=       <Digit> | '#' | '*'

Digit ::=       '0' | '1' | '2' | '3' | '4'
               | '5' | '6' | '7' | '8' | '9'

Pause ::=         ','
```

After calling this function, sub-launched applications can receive notifications when the following telephony events occur. Note that these notifications can be raised after both synchronous and asynchronous calls to this function.

Table 76.2 Telephony events sent to sublaunched applications

Event	Description
`sysTelSpcLaunchCmdCallReleased`	This is passed to a sub-launched application to warn that the telephone call has been released.
`sysTelSpcLaunchCmdCallBusy`	This is passed to a sub-launched application to warn that the called equipment is busy.

Table 76.2 Telephony events sent to sublaunched applications

Event	Description
`sysTelSpcLaunchCmdCallConnect`	This is passed to a sub-launched application to warn that the line is open. The ID of the open line is stored in the `UInt32` value of the parameter block passed to the application.
`sysTelSpcLaunchCmdCallError`	This is passed to a sub-launched application to warn that the telephone call encountered an error.

Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSpcCloseLine()`

TelSpcCloseLine Function

Purpose Ends a voice telephone call.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSpcCloseLine(UInt16 iRefnum,
    TelAppID iAppId, UInt8 iLineId,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

→ *iLineId*
The ID of the voice line that you want to close. This is the ID returned by a previous call to the `TelSpcAcceptCall()` function.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `iLineId` parameter.

`functionId`
: `kTelSpcCloseLineMessage`

Comments

Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelSpcCallNumber()`

TelSpcConference Function

Purpose

Initiate a conference telephone call by merging the active line and the held line.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelSpcConference(UInt16 iRefnum,
   TelAppID iAppId, UInt8 *oLineIdP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*

The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

← *oLineIdP*
: A pointer to an unsigned byte value. Upon return, this is the ID of the voice line assigned to the telephone call.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `oLineIdP` parameter.

`functionId`
: `kTelSpcConferenceMessage`

WARNING! When using this function asynchronously, you must ensure that the value referenced by `oLineIdP` remains in memory until the asynchronous call completes.

Comments

Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSpcCallNumber()`, `TelSpcCloseLine()`, `TelSpcHoldLine()`, `TelSpcRetrieveHeldLine()`, `TelSpcSelectLine()`

TelSpcGetCallerNumber Function

Purpose Retrieve the telephone number of the caller on an incoming telephone call.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSpcGetCallerNumber(UInt16 iRefnum,
    TelAppID iAppId,
    TelSpcGetCallerNumberType *ioParamP,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioParamP*
A pointer to a `TelSpcGetCallerNumberType` structure that is used to retrieve the caller's telephone number.

On input, the `size` field of this structure specifies the allocated size of the `value` buffer. Upon return, the `size` field specifies the actual size of the telephone number, even if it was truncated to fit into the buffer.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful. If there is no active incoming telephone call, this function returns the `telErrUnavailableValue` error.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelSpcGetCallerNumberType` structure passed to this function in the `ioParamP` parameter.

`functionId`
: `kTelSpcGetCallerNumberMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments

The emergency telephone call number is stored into the `value` field of the `TelSpcGetCallerNumberType` structure referenced by `ioParamP`. If the `value` buffer is too small to contain the complete string, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `size` field of the structure is always updated to contain the actual size of the complete string.

Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelSpcAcceptCall()`

TelSpcHoldLine Function

Purpose

Put the currently active voice line on hold.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelSpcHoldLine(UInt16 iRefnum,
   TelAppID iAppId, UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: A `NULL` pointer.

`functionId`
: `kTelSpcHoldLineMessage`

Comments

Note that there can only be one line active at any given time, and there can only be one line on hold at any given time.

Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSpcRetrieveHeldLine()`

TelSpcPlayDTMF Function

Purpose Play a dual-tone multi-frequency sound to the network system for a specified duration. Note that you can only play a DTMF while a voice telephone call is active.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSpcPlayDTMF(UInt16 iRefnum,
   TelAppID iAppId, TelSpcPlayDTMFType *iParamP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

→ *iParamP*
A pointer to a `TelSpcPlayDTMFType` structure that specifies the tone to play and its duration.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
`errNone` upon success or an error code upon failure.

`transId`
The transaction ID of the operation.

`paramP`
: Points to the `TelSpcPlayDTMFType` structure passed to this function in the `iParamP` parameter

`functionId`
: `kTelSpcPlayDTMFMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `iParamP` remains in memory until the asynchronous call completes.

Comments Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSpcSendBurstDTMF()`, `TelSpcStartContinuousDTMF()`, `TelSpcStopContinuousDTMF()`

TelSpcRejectCall Function

Purpose Reject an incoming voice telephone call.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSpcRejectCall(UInt16 iRefnum,
   TelAppID iAppId, UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: A `NULL` pointer.

`functionId`
: `kTelSpcRejectCallMessage`

Comments

Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelSpcAcceptCall()`, `TelSpcGetCallerNumber()`

TelSpcRetrieveHeldLine Function

Purpose

Reconnect the voice line that is currently on hold, making it the active voice line.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelSpcRetrieveHeldLine(UInt16 iRefnum,
   TelAppID iAppId, UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: A `NULL` pointer.

`functionId`
: `kTelSpcRetrieveHeldLineMessage`

Comments Note that there can only be one line active at any given time, and there can only be one line on hold at any given time.

Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSpcHoldLine()`

TelSpcSelectLine Function

Purpose Select the specified line ID as the newly active voice line.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSpcSelectLine(UInt16 iRefnum,
   TelAppID iAppId, UInt8 iLineId,
   UInt16 *ioTransIdP)
```

Parameters

→ *`iRefnum`*
: The telephony manager library reference number.

→ *`iAppId`*
: The telephone application attachment identifier for your application.

→ *iLineId*
: The ID of the voice line that you want to activate. This is the ID returned by a previous call to the `TelSpcAcceptCall()` function.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `iLineId` parameter.

`functionId`
: `kTelSpcSelectLineMessage`

Comments

If a line was active previous to completion of this function, that line is put on hold. Note that there can only be one line active at any given time, and there can only be one line on hold at any given time.

Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelSpcConference()`

TelSpcSendBurstDTMF Function

Purpose

Send a string of dual-tone, multi-frequency sounds to the network system. Note that you can only play a DTMF while a voice telephone call is active.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelSpcSendBurstDTMF(UInt16 iRefnum,
   TelAppID iAppId, const Char *iDTMFStringP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

→ *iDTMFStringP*
: A null-terminated string of keytone values. Each byte of the string specifies one of the values declared under "Keycode Constants" on page 1669.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the string passed to this function in the `iDTMFStringP` parameter.

```
functionId
     kTelSpcSendBurstDTMFMessage
```

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `iDTMFStringP` remains in memory until the asynchronous call completes.

Comments This function sends a burst string of keytones to the network. Each key tone is played for the default duration defined by the network.

Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSpcPlayDTMF()`, `TelSpcStartContinuousDTMF()`, `TelSpcStopContinuousDTMF()`

TelSpcStartContinuousDTMF Function

Purpose Send a continuous dual-tone, multi-frequency sound to the network system. Note that you can only play a DTMF while a voice telephone call is active.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSpcStartContinuousDTMF(UInt16 iRefnum,
   TelAppID iAppId, UInt8 iKeyCode,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

→ *iKeyCode*
The keycode to send to the network. This must be one of the values declared under "Keycode Constants" on page 1669.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `iKeyCode` parameter.

`functionId`
: `kTelSpcStartContinuousDTMFMessage`

Comments

This function sends a key tone to the network system that is played continuously until the `TelSpcStopContinuousDTMF()` function executes.

Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelSpcPlayDTMF()`, `TelSpcSendBurstDTMF()`, `TelSpcStopContinuousDTMF()`

TelSpcStopContinuousDTMF Function

Purpose

Stop the continuous playing of a tone that was started by calling the `TelSpcStartContinuousDTMF()` function.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelSpcStopContinuousDTMF(UInt16 iRefnum,
    TelAppID iAppId, UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: A `NULL` pointer.

`functionId`
: `kTelSpcStopContinuousDTMFMessage`

Comments

This function stops the continuous playing of the tone that was previously initiated by calling the `TelSpcStartContinuousDTMF()` function.

Before using this function, you should verify that it is available by calling the `TelIsSpcServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSpcPlayDTMF()`, `TelSpcSendBurstDTMF()`, `TelSpcStartContinuousDTMF()`

77

Telephony SMS

This chapter describes the telephony SMS service set of the telephony API.

For more information about the telephony manager basic services and the different service sets, see Chapter 73, "Telephony Basic Services."

This chapter describes:

- Telephony SMS Data Structures
- Telephony SMS Constants
- Telephony SMS Functions

For more information about using the telephony manager, see Chapter 10, "Telephony Manager," in *Palm OS Programmer's Companion*, vol. II, *Communications*.

Telephony SMS Data Structures

This section describes the data structures used with the SMS service set of the telephony API.

TelSmsDateTimeType Struct

Purpose Several of the other data structures used with the Telephony SMS functions include an `TelSmsDateTimeType` structure to store a date and time value.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsDateTimeType {
   Boolean absolute;
   UInt32 dateTime;
} TelSmsDateTimeType
```

Fields

→ `absolute`
If `true`, the `dateTime` value is an absolute time value specified as the number of seconds since 1/1/1904. This is the format returned by the `TimGetSeconds()` function. If `false`, the `dateTime` value is relative to the current date and time.

→ `dateTime`
The date and time value.

TelSmsDeleteMessageType Struct

Purpose

The `TelSmsDeleteMessage()` function uses a `TelSmsDeleteMessageType` structure to specify the message to be deleted.

Declared In

`TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsDeleteMessageType {
   UInt8 messageType;
   UInt16 index;
} TelSmsDeleteMessageType
```

Fields

→ `messageType`
The message type. This is one of the SMS Message Type Constants.

→ `index`
The index of the SMS message in the phone's storage that is to be deleted. Note that the message is deleted from the storage area selected by a call to the `TelSmsSelectStorage()` function.

Note that the index is zero-based.

TelSmsDeliveryAdvancedCDMAType Struct

Purpose The `TelSmsDeliveryMessageType` structure includes a `TelSmsDeliveryAdvancedCDMAType` structure for CDMA messages.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsDeliveryAdvancedCDMAType {
   UInt8 messageType;
   TelSmsDateTimeType validityPeriod;
   UInt8 priority;
   UInt8 privacy;
   Boolean alertOnDeliveryRequest;
   Boolean manualAckRequest;
   UInt8 voiceMessageNumber;
   UInt8 callbackNumberSize;
   Char *callbackNumberAddress;
   UInt8 languageIndicator;
} TelSmsDeliveryAdvancedCDMAType
```

Fields

`messageType`
: The type of the message. This must be one of the SMS Message Type Constants.

`validityPeriod`
: An `TelSmsDateTimeType` structure that specifies the amount of time for which the message is valid.

`priority`
: The message priority. This must be one of the SMS Message Urgency Constants.

`privacy`
: The privacy type of the message. This must be one of the SMS Message Privacy Constants.

`alertOnDeliveryRequest`
: `true` if the user is to be alerted upon delivery of this message, and `false` if not.

`manualAckRequest`
: `true` if a reply is requested from the user, and `false` if not.

`voiceMessageNumber`
: The index of a new message in the in box.

`callbackNumberSize`
: When the structure is used as an input parameter, this is the allocated size, in bytes, of the `callbackNumberAddress` string.

  Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

`callbackNumberAddress`
: A buffer into which the callback number address string is stored.

  Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a `telErrBufferSize` error.

`languageIndicator`
: Reserved.

TelSmsDeliveryAdvancedGSMType Struct

Purpose The `TelSmsDeliveryMessageType` structure includes a `TelSmsDeliveryAdvancedGSMType` structure for GSM messages.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsDeliveryAdvancedGSMType {
   UInt16 protocolId;
   Boolean replyPath;
   Char *serviceCenterNumber;
   UInt8 serviceCenterNumberSize;
} TelSmsDeliveryAdvancedGSMType
```

Fields

← `protocolId`
: The protocol used for this message. This is one of the SMS Message Transport Protocol Constants.

← `replyPath`

If this value is set, then you use the `serviceCenterNumber` to reply to this message.

If this value is not set, you use the default service center provided by your network operator.

↔ `serviceCenterNumber`

A buffer into which the service center number string is stored.

Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a `telErrBufferSize` error.

↔ `serviceCenterNumberSize`

When the structure is used as an input parameter, this is the allocated size, in bytes, of the `serviceCenterNumber` string.

Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

TelSmsDeliveryAdvancedTDMAType Struct

Purpose The `TelSmsDeliveryMessageType` structure includes a `TelSmsDeliveryAdvancedTDMAType` structure for TDMA messages.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsDeliveryAdvancedTDMAType {
   UInt8 messageType;
   TelSmsDateTimeType validityPeriod;
   UInt8 priority;
   UInt8 privacy;
   Boolean manualAckRequest;
```

```
   Boolean alertOnDeliveryRequest;
   UInt8 voiceMessageNumber;
   UInt8 callbackNumberSize;
   Char *callbackNumberAddress;
} TelSmsDeliveryAdvancedTDMAType
```

Fields

`messageType`
: The type of the message. This must be one of the SMS Message Type Constants.

`validityPeriod`
: An `TelSmsDateTimeType` structure that specifies the amount of time for which the message is valid.

`priority`
: The message priority. This must be one of the SMS Message Urgency Constants.

`privacy`
: The privacy type of the message. This must be one of the SMS Message Privacy Constants.

`alertOnDeliveryRequest`
: `true` if the user is to be alerted upon delivery of this message, and `false` if not.

`manualAckRequest`
: `true` if a reply is requested from the user, and `false` if not.

`voiceMessageNumber`
: The index of the message in the in box.

`callbackNumberSize`
: When the structure is used as an input parameter, this is the allocated size, in bytes, of the `callbackNumberAddress` string.

  Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

`callbackNumberAddress`

A buffer into which the callback telephone number address string is stored.

Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a `telErrBufferSize` error.

TelSmsDeliveryMessageType Struct

Purpose The `TelSmsReadMessage()` function uses a `TelSmsDeliveryMessageType` structure to retrieve information about a delivered message.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsDeliveryMessageType {
   UInt16 version;
   UInt16 index;
   UInt32 messageIdentifier;
   TelSmsDateTimeType timeStamp;
   UInt16 dataSize;
   UInt8 *data;
   UInt8 dataCodingScheme;
   UInt8 originatingAddressSize;
   Char *originatingAddress;
   Boolean otherToReceive;
   Boolean reportDeliveryIndicator;
   UInt8 standardType;
   union {
      TelSmsDeliveryAdvancedGSMType advancedGSM;
      TelSmsDeliveryAdvancedCDMAType advancedCDMA;
      TelSmsDeliveryAdvancedTDMAType advancedTDMA;
   } advancedParams;
   UInt8 extensionsCount;
   TelSmsExtensionType *extensionsP;
} TelSmsDeliveryMessageType
```

Fields

→ `version`

The version of the SMS API associated with this message.

↔ `index`

Upon return, the SMS index of the message on the phone. This is a 0-based index.

This value is used for input only when calling the `TelSmsReadMessage()` function to read one SMS at a time.

← `messageIdentifier`

The message identifier.

← `timeStamp`

The message time stamp. This is a `TelSmsDateTimeType` structure.

↔ `dataSize`

The size of the `data` buffer.

When the structure is used as an input parameter, this is the allocated size of the `data` buffer.

Upon return, this is the actual size of the message data. If the buffer is too small to contain the entire message, this field is assigned the entire length of the message, and the function using this structure generates a `telErrBufferSize` error.

↔ `data`

A buffer into which the retrieved data is stored.

Note that if this buffer is too small to contain the entire retrieved message, the end of the message is truncated and the function using this structure generates a `telErrBufferSize` error.

← `dataCodingScheme`

The coding scheme used for the data. This is one of the SMS Data Coding Scheme Constants.

↔ `originatingAddressSize`

The size of the `originatingAddress` buffer.

When the structure is used as an input parameter, this is the allocated size of the `originatingAddress` buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

↔ `originatingAddress`
: A buffer into which the originating address string is stored.

  Note that if this buffer is too small to contain the entire string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a `telErrBufferSize` error.

← `otherToReceive`
: Indicates whether there are more messages waiting to be received from the service center to the mobile device.

← `reportDeliveryIndicator`
: If `true`, indicates that the originating user has asked the network to send a delivery report.

← `standardType`
: Indicates which field of the `advancedParams` union contains the message information. This is one of the Network Type Constants described in Chapter 75, "Telephony Network."

  If this value is `kTelNwkCDMA`, then the `advancedParams` union contains a `TelSmsDeliveryAdvancedCDMAType` structure, and similarly for the other values.

`advancedParams`
: Advanced message information for GSM, CDMA, or TDMA messages. This is one of the following structure types:

  `TelSmsDeliveryAdvancedGSMType`

  `TelSmsDeliveryAdvancedCDMAType`

  `TelSmsDeliveryAdvancedTDMAType`

↔ `extensionsCount`
: On input, this is the number of extension structures allocated for this message. You must allocate at least one structure to specify the multipart information.

  Upon return, this is the number of extensions in the SMS header. If the SMS header contains more extensions than you have allocated, the available extension structures are filled, and this function generates a `telErrBufferSize` error.

↔ `extensionsP`
A pointer to an array of `TelSmsExtensionType` structures that you have allocated for this message. You must allocate this array before using this structure.

TelSmsExtensionType Struct

Purpose The `TelSmsExtensionType` structure specifies multipart information about a message.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsExtensionType {
   UInt8 extensionTypeId;
   union {
      TelSmsMultiPartExtensionType mp;
      TelSmsNbsExtensionType nbs;
      TelSmsUserExtensionType user;
   } extension;
} TelSmsExtensionType
```

Fields ↔ `extensionTypeId`
Identifies the type of SMS extension structure found in the `extension` union. This is one of the SMS Extension Type Constants.

If the value of this field is `kTelSmsMultipartExtensionTypeId`, then the extension union contains a `TelSmsMultiPartExtensionType` structure.

If the value of this field is `kTelSmsNbsExtensionTypeId` or `kTelSmsNbs2ExtensionTypeId`, then the union contains a `TelSmsNbsExtensionType` structure. The difference between these two is that `kTelSmsNbs2ExtensionTypeId` indicates that the port value is a long instead of a short.

If this field contains any other value, then the union contains a `TelSmsUserExtensionType` structure.

← `extension`

The extension information, which is one of the following structure types:

`TelSmsMultiPartExtensionType`

`TelSmsNbsExtensionType`

`TelSmsUserExtensionType`

TelSmsGetAvailableStorageType Struct

Purpose The `TelSmsGetAvailableStorage()` function uses a `TelSmsGetAvailableStorageType` structure to retrieve information about the storage available on the phone.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsGetAvailableStorageType {
   UInt16 count;
   UInt8 *storagesP;
} TelSmsGetAvailableStorageType
```

Fields

↔ `count`

The size of the `storagesP` buffer.

When the structure is used as an input parameter, this is the allocated number of values in the buffer. Upon return, this is the total number of storage areas in the phone.

← `storagesP`

A buffer into which the retrieved storage IDs are stored. Each value stored into the buffer is one of the SMS Storage ID Constants.

TelSmsGetMessageCountType Struct

Purpose The `TelSmsGetMessageCount()` function uses a `TelSmsGetMessageCountType` structure to retrieve information about messages in the currently selected storage.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsGetMessageCountType {
   UInt8 messageType;
   UInt16 slots;
   UInt16 count;
} TelSmsGetMessageCountType
```

Fields

→ `messageType`
: The type of message for which you want to retrieve the count. This must be one of the SMS Message Type Constants.

  You must fill this in on input to the `TelSmsGetMessageCount()` function.

← `slots`
: The total number of message slots available in the currently selected storage area (for all message types).

← `count`
: The number of filled slots for the specified `messageType` in the currently selected storage area.

TelSmsManualAckType Struct

Purpose The `TelSmsSendManualAcknowledge()` function uses a `TelSmsManualAckType` structure to specify the information used to send a message acknowledgment.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsManualAckType {
   UInt16 version;
   Char *destinationAddress;
   UInt32 messagesId;
   UInt16 dataSize;
   UInt8 *data;
   UInt8 dataCodingScheme;
   UInt8 responseCode;
} TelSmsManualAckType
```

Fields

`version`
The version of the SMS API used for this acknowledgment.

`destinationAddress`
The destination address. For GSM, the length of this address is 12 bytes. For CDMA, the length is up to 128 bytes.

`messageId`
Upon return, the ID of the acknowledgment.

`dataSize`
The size of the `data` buffer.

When the structure is used as an input parameter, this is the allocated size of the `data` buffer.

Upon return, this is the actual size of the data. If the buffer is too small to contain all of the data, this field is assigned the entire length of the data, and the function using this structure generates a `telErrBufferSize` error.

`data`
A buffer into which the retrieved data is stored.

Note that if this buffer is too small to contain the all of the retrieved data, the data is truncated and the function using this structure generates a `telErrBufferSize` error.

`dataCodingScheme`
The coding scheme used for the data. This must be one of the SMS Data Coding Scheme Constants.

`responseCode`
The response code. The value of this field depends on the network being used.

TelSmsMultiPartExtensionType Struct

Purpose The `TelSmsExtensionType` structure uses a `TelSmsMultiPartExtensionType` structure to describe information about a multipart message.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsMultiPartExtensionType {
   UInt16 bytesSent;
   UInt16 partCurrent, partCount, partId;
} TelSmsMultiPartExtensionType
```

Fields

↔ `bytesSent`

On input, set this value to 0.

Upon return, this is the current count of message bytes that have been sent.

↔ `partCurrent`

On input, set this value to 0.

Upon return, this is the part number of the current message part.

↔ `partCount`

On input, set this value to 0.

Upon return, this is the number of message parts required to send the data.

↔ `partId`

The ID of the current SMS message. This ID is unique and is the same for all parts of the message. This information is required to reassemble a multi-part SMS.

On input, set this value to 0.

TelSmsNbsExtensionType Struct

Purpose The `TelSmsExtensionType` structure uses a `TelSmsNbsExtensionType` structure to describe information about a Narrow Band Sockets (NBS) message.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsNbsExtensionType {
   UInt16 destPort, srcPort;
} TelSmsNbsExtensionType
```

Fields

↔ `destPort`

When the structure is used for input, this is the NBS port number used to encode the data.

Upon return, this is the NBS port number that was used for the data.

↔ `srcPort`

This is currently the same as the `destPort`.

TelSmsReadMessagesType Struct

Purpose The `TelSmsReadMessages()` function uses a `TelSmsReadMessagesType` structure to retrieve messages from the currently selected storage area.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsReadMessagesType {
   UInt16 first;
   UInt16 count;
   TelSmsDeliveryMessageType *messagesP;
} TelSmsReadMessagesType
```

Fields

→ `first`

The index of the first message to retrieve. Message indexes are zero-based.

↔ `count`

The size of the `messagesP` buffer.

When the structure is used as an input parameter, this is the allocated number of pointers in the `messagesP` buffer.

Upon return, this is the number of messages that could be retrieved. If the `messagesP` buffer is too small to contain all

of the messages, this field is assigned the entire count, and the function using this structure generates a `telErrBufferSize` error.

↔ `messagesP`
An array of pointers to `TelSmsDeliveryMessageType` structures, each of which is filled in with a retrieved message, if available.

TelSmsReadReportsType Struct

Purpose The `TelSmsReadReports()` function uses a `TelSmsReadReportsType` structure to retrieve reports from the currently selected storage area.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsReadReportsType {
   UInt16 first;
   UInt16 count;
   TelSmsReportType *reportsP;
} TelSmsReadReportsType
```

Fields

→ `first`
The index of the first report to retrieve. Report indexes are zero-based.

↔ `count`
The size of the `reportsP` buffer.

When the structure is used as an input parameter, this is the allocated number of pointers in the `reportsP` buffer.

Upon return, this is the actual number of reports that could be read. If the buffer is too small to contain all of the reports, this field is assigned the entire count, and the function using this structure generates a `telErrBufferSize` error.

↔ `reportsP`
An array of pointers to `TelSmsReportType` structures, each of which is filled in with a retrieved message, if available.

Note that if this buffer is too small to contain all of the retrieved reports, the function using this structure generates a `telErrBufferSize` error.

TelSmsReadSubmittedMessagesType Struct

Purpose The `TelSmsReadSubmittedMessages()` function uses a `TelSmsReadSubmittedMessagesType` structure to retrieve submitted messages from the currently selected storage area.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsReadSubmittedMessagesType {
   UInt16 first;
   UInt16 count;
   TelSmsSubmittedMessageType *submittedsP;
} TelSmsReadSubmittedMessagesType
```

Fields

→ `first`
: The index of the first message to retrieve. Message indexes are zero-based.

↔ `count`
: The size of the `submittedsP` buffer.

  When the structure is used as an input parameter, this is the allocated number of pointers in the `submittedsP` buffer. Upon return, this is the number of messages that were actually read.

↔ `submittedsP`
: An array of pointers to `TelSmsSubmittedMessageType` structures, each of which is filled in with a retrieved message, if available.

TelSmsReportType Struct

Purpose The `TelSmsReadReport()` function uses a `TelSmsReportType` structure to retrieve a report from the report storage area.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsReportType {
   UInt16 version;
   UInt16 index;
   UInt8 reportType;
   UInt32 messageId;
   UInt16 dataSize;
   UInt8 *data;
   UInt8 dataCodingScheme;
```

```
    Char *originatingAddress;
    UInt8 originatingAddressSize;
    UInt8 report;
    TelSmsDateTimeType timeStamp;
} TelSmsReportType
```

Fields

→ `version`
: The version of the SMS API. The current version number is `kTelSmsAPIVersion`.

← `index`
: The index of the report in the phone storage area.

← `reportType`
: The delivery report type. This is one of the SMS Report Types.

← `messageId`
: The message ID.

↔ `dataSize`
: The size of the `data` buffer.

  When the structure is used as an input parameter, this is the allocated size of the `data` buffer.

  Upon return, this is the actual size of the data. If the buffer is too small to contain all of the data, this field is assigned the entire length of the data, and the function using this structure generates a `telErrBufferSize` error.

↔ `data`
: A buffer into which the retrieved data is stored.

  Note that if this buffer is too small to contain the all of the retrieved data, the data is truncated and the function using this structure generates a `telErrBufferSize` error.

← `dataCodingScheme`
: The encoding scheme used for the report data. This is one of the SMS Data Coding Scheme Constants.

↔ `originatingAddress`
: A buffer into which the originating address is stored.

  Note that if this buffer is too small to contain the entire retrieved string, the string is truncated (and ends with the null terminator character) and the function using this structure generates a `telErrBuffrSize` error.

↔ `originatingAddressSize`
The size of the `originatingAddress` buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

← `report`
The ID of the delivery confirmation report associated with the message. This is one of the SMS Delivery Status Reports constants.

← `timeStamp`
An `TelSmsDateTimeType` structure that stores the time at which the message corresponding to the report was received.

TelSmsSendMessageType Struct

Purpose The `TelSmsSendMessage()` function uses this structure to send an SMS message.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsSendMessageType {
   UInt32 messageId;
   TelSmsSubmitMessageType message;
} TelSmsSendMessageType
```

Fields

← `messageId`
The SMS ID that was assigned by the telephone to the outgoing message.

For a multi-part message, each part has its own message ID.

→ `message`
A structure of type `TelSmsSubmitMessageType` that contains the message data and parameters.

TelSmsSubmitAdvancedCDMAType Struct

Purpose The `TelSmsSubmitMessageType` structure includes a `TelSmsSubmitAdvancedCDMAType` structure for CDMA messages.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsSubmitAdvancedCDMAType {
   Boolean manualAckRequest;
   UInt8 messageType;
   TelSmsDateTimeType deferredDate;
   UInt8 priority;
   UInt8 privacy;
   Boolean alertOnDeliveryRequest;
   Char *callbackNumber;
   UInt8 callbackNumberSize;
} TelSmsSubmitAdvancedCDMAType
```

Fields

`manualAckRequest`
: `true` if a reply is requested from the user, and `false` if not.

`messageType`
: The type of the message. This must be one of the SMS Message Type Constants.

`deferredDate`
: A date and time at which the message should be delivered. See `TelSmsDateTimeType`.

`priority`
: The message priority. This must be one of the SMS Message Urgency Constants.

`privacy`
: The privacy type of the message. This must be one of the SMS Message Privacy Constants.

`alertOnDeliveryRequest`
: `true` if the user is to be alerted upon delivery of this message, and `false` if not.

`callbackNumber`
: A buffer into which the retrieved callback telephone number string is stored.

  Note that if this buffer is too small to contain the entire retrieved string, the string is truncated (and ends with the

null terminator character) and the function using this structure generates a `telErrBufferSize` error.

`callbackNumberSize`
When the structure is used as an input parameter, this is the allocated size, in bytes, of the `callbackNumber` string.

Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

TelSmsSubmitAdvancedGSMType Struct

Purpose The `TelSmsSubmitMessageType` structure includes a `TelSmsSubmitAdvancedGSMType` structure for GSM messages.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsSubmitAdvancedGSMType {
   UInt16 protocolId;
   Boolean rejectDuplicatedRequest;
   Boolean replyPath;
   Char *serviceCenterNumber;
   UInt8 serviceCenterNumberSize;
} TelSmsSubmitAdvancedGSMType
```

Fields

→ `protocolId`
The protocol used for this message. This is one of the SMS Message Transport Protocol Constants.

→ `rejectDuplicatedRequest`
A Boolean value that specifies if the service center should accept a submit message for a submit message that is still held in the service center when that message has the same identifier and destination address as a previously submitted message from the same originating address.

→ `replyPath`
`true` if the next two parameters are set. `false` otherwise.

→ `serviceCenterNumber`
A buffer containing the SMS center number string. Replies to the originating message should go through this service center. Normally, you should set this to `NULL`.

→ `serviceCenterNumberSize`
The allocated size, in bytes, of the `serviceCenterNumber` string. Normally, the string is `NULL` and its size is 0.

TelSmsSubmitAdvancedTDMAType Typedef

Purpose The `TelSmsSubmitMessageType` structure includes a `TelSmsSubmitAdvancedTDMAType` structure for TDMA messages.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef TelSmsSubmitAdvancedCDMAType
TelSmsSubmitAdvancedTDMAType
```

TelSmsSubmitMessageType Struct

Purpose The `TelSmsReadReports()` and `TelSmsSendMessageType` structures use a `TelSmsSubmitMessageType` structure to stored reports retrieved from the currently selected storage area.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsSubmitMessageType {
   UInt16 version;
   Boolean networkDeliveryRequest;
   Char *destinationAddress;
   UInt8 destinationAddressSize;
   UInt16 dataSize;
   UInt8 *data;
   UInt8 dataCodingScheme;
   TelSmsDateTimeType validityPeriod;
   UInt8 standardType;
   union {
      TelSmsSubmitAdvancedGSMType advancedGSM;
      TelSmsSubmitAdvancedCDMAType advancedCDMA;
      TelSmsSubmitAdvancedTDMAType advancedTDMA;
   } advancedParams;
   UInt8 extensionsCount;
   TelSmsExtensionType *extensionsP;
} TelSmsSubmitMessageType
```

Fields

→ `version`
The version of the SMS API associated with this message. The constant `kTelSmsAPIVersion` gives the current version number.

→ `networkDeliveryRequest`
If this value is `true`, an acknowledgement is required.

↔ `destinationAddress`
A buffer that contains the phone number of the message recipient.

↔ `destinationAddressSize`
The size of the destination address string.

↔ `dataSize`
The size of the `data` buffer.

↔ `data`
A buffer into which the retrieved message data is stored. Note that \0 is a meaningful character in the SMS character set and does not imply the end of the string.

→ `dataCodingScheme`
The coding scheme used for the data. This is one of the SMS Data Coding Scheme Constants.

→ `validityPeriod`
An `TelSmsDateTimeType` structure that specifies the amount of time for which the message is valid.

→ `standardType`
Indicates which field of the `advancedParams` union contains the message information. This must be one of the Network Type Constants described in Chapter 75, "Telephony Network."

If this value is `kTelNwkCDMA`, then the `advancedParams` union contains a `TelSmsDeliveryAdvancedCDMAType` structure, and similarly for the other values.

→ `advancedParams`
Advanced message information for GSM, CDMA, or TDMA messages. This is a pointer to one of the following structure types:

`TelSmsSubmitAdvancedGSMType`

`TelSmsSubmitAdvancedCDMAType`

`TelSmsSubmitAdvancedTDMAType`

↔ `extensionsCount`
The number of extension structures allocated for this message. You must allocate at least one structure to specify the multipart information.

↔ `extensionsP`
A pointer to an array of `TelSmsExtensionType` structures that you have allocated for this message.

TelSmsSubmittedMessageType Struct

Purpose The `TelSmsReadSubmittedMessage()` function uses a `TelSmsSubmittedMessageType` structure to retrieve reports from the currently selected storage area.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsSubmittedMessageType {
   UInt16 index;
   TelSmsSubmitMessageType message;
} TelSmsSubmittedMessageType
```

Fields

→ `index`
The index of the message on the phone.

← `message`
A `TelSmsSubmitMessageType` structure that represents the message.

TelSmsUserExtensionType Struct

Purpose The `TelSmsExtensionType` structure uses a `TelSmsUserExtensionType` structure to describe a user-defined extended message header.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelSmsUserExtensionType {
   UInt8 *extHeader;
   UInt8 extHeaderSize;
} TelSmsUserExtensionType
```

Fields

↔ `extHeader`
On input, this field must be set to 0.

Upon return, this is a pointer to the user-defined header content.

↔ `extHeaderSize`
On input, this field must be set to 0.

Upon return, this is the size of the user-defined header content.

Telephony SMS Constants

This section describes the constants used with the SMS service set of the telephony API.

SMS Extension Type Constants

Purpose The SMS extension type constants describe the type of extension used to represent part of a message. You can use one of the types listed here or define your own.

Declared In `TelephonyMgr.h`

Constants

`#define kTelSmsMultiPartExtensionTypeId 0x00`
A multipart short message.

`#define kTelSmsNbs2ExtensionTypeId 0x05`
An NBS message with long port number value.

`#define kTelSmsNbsExtensionTypeId 0x04`
An NBS message with short port number value.

SMS Message Type Constants

Purpose The SMS message type constants describe the delivery type of a message.

Declared In `TelephonyMgr.h`

`#define kTelSmsMessageTypeDelivered 0`
A delivered message.

`#define kTelSmsMessageTypeReport 1`
A report message.

`#define kTelSmsMessageTypeSubmitted 2`
A submitted message.

`#define kTelSmsMessageTypeManualAck 3`
A manual acknowledgement message.

`#define kTelSmsMessageAllTypes 4`
All messages.

SMS Message Transport Protocol Constants

Purpose The SMS message transport protocol constants describe the protocol used to deliver a message.

Declared In `TelephonyMgr.h`

Constants `#define kTelSmsDefaultProtocol 0`
The default message transport protocol.

`#define kTelSmsFaxProtocol 1`
A FAX message.

`#define kTelSmsX400Protocol 2`
An X.400 message.

`#define kTelSmsPagingProtocol 3`
A paged message.

`#define kTelSmsEmailProtocol 4`
An email message.

`#define kTelSmsErmesProtocol 5`
An Ermes message.

`#define kTelSmsVoiceProtocol 6`
A voice message.

SMS Storage ID Constants

Purpose The SMS storage ID constants describe the storage location of a message.

Declared In `TelephonyMgr.h`

Constants

`#define kTelSmsStorageSIM 0`
Stored in the SIM.

`#define kTelSmsStoragePhone 1`
Stored in the phone.

`#define kTelSmsStorageAdaptor 2`
Stored in the telephone adaptor.

`#define kTelSmsStorageFirstOem 3`
Storage managed by the OEM.

This constant specifies the first OEM storage area. You can specify additional OEM storage areas by incrementing this value. For example, to specify the third OEM storage area, use `kTelSmsStorageFirstOem+2`.

SMS Data Coding Scheme Constants

Purpose The SMS data coding scheme constants describe the encoding used for SMS data.

Declared In `TelephonyMgr.h`

`#define kTelSms8BitsEncoding 0`
8-bit encoding.

`#define kTelSmsBitsASCIIEncoding 1`
ANSI X3.4 encoding.

`#define kTelSmsIA5Encoding 2`
CCITT T.50 encoding.

`#define kTelSmsIS91Encoding 3`
TIA/EIA/IS-91 section 3.7.1 encoding.

`#define kTelSmsUCS2Encoding 4`
UCS2 encoding; used with GSM only.

`#define kTelSmsDefaultGSMEncoding 5`
Default encoding for GSM only.

SMS Message Urgency Constants

Purpose The SMS message urgency constants describe the priority level of a message in a `TelSmsDeliveryAdvancedCDMAType` or `TelSmsDeliveryAdvancedTDMAType` structure.

Declared In `TelephonyMgr.h`

Constants

`#define kTelSmsUrgencyNormal 0`
Normal urgency.

`#define kTelSmsUrgencyUrgent 1`
An urgent message.

`#define kTelSmsUrgencyEmergency 2`
An emergency message.

SMS Message Privacy Constants

Purpose The SMS message privacy constants describe the privacy type of a message in a CDMA or TDMA advanced parameters.

Declared In `TelephonyMgr.h`

Constants

`#define kTelSmsPrivacyNotRestricted 0`
Privacy level 0.

`#define kTelSmsPrivacyRestricted 1`
Privacy level 1.

`#define kTelSmsPrivacyConfidential 2`
Privacy level 2.

`#define kTelSmsPrivacySecret 3`
Privacy level 3.

SMS Message Types

Purpose

Declared In `TelephonyMgr.h`

Constants

`#define kTelSmsCMTMessageType 0`
Cellular Messaging Teleservice (CMT) message.

`#define kTelSmsCPTMessageType 1`
Cellular Paging Teleservice (CPT) message type.

`#define kTelSmsVMNMessageType 2`
Voice Mail Notification (VMN) message type.

SMS Delivery Status Reports

Purpose Delivery status report codes used in the `report` field of the `TelSmsReportType` structure.

Declared In `TelephonyMgr.h`

Constants

`#define kTelSmsDSRSuccess 0`
Success.

`#define kTelSmsDSRMessageReplaced 1`
Message replaced.

`#define kTelSmsDSRMessageForwarded 2`
Message forwarded.

`#define kTelSmsDSRTempCongestion 3`
Temporarily not delivered due to congestion.

`#define kTelSmsDSRTempSMEBusy 4`
Temporarily not delivered due to the mobile phone being busy.

`#define kTelSmsDSRTempServiceRejected 5`
Temporarily not delivered because the service rejected the message.

`#define kTelSmsDSRTempServiceUnavailable 6`
Temporarily not delivered due to the service being unavailable.

`#define kTelSmsDSRTempSMEError 7`
Temporarily not delivered due to an error in the mobile phone.

`#define kTelSmsDSRTempOther 8`
Temporarily not delivered due to some other cause.

`#define kTelSmsDSRPermRPError 9`
Delivery failed due to a reply path error.

`#define kTelSmsDSRPermBadDestination 10`
Delivery failed due to a bad destination address.

`#define kTelSmsDSRPermUnobtainable 11`
Delivery failed due to an error.

`#define kTelSmsDSRPermServiceUnavailable 12`
Delivery failed due to service unavailability.

`#define kTelSmsDSRPermInternetworkError 13`
Delivery failed due to an networking error.

`#define kTelSmsDSRPermValidityExpired 14`
Delivery failed due to its validity expiring.

`#define kTelSmsDSRPermDeletedByOrigSME 15`
Delivery failed due to the message being deleted by the originating mobile phone.

`#define kTelSmsDSRPermDeleteByAdm 16`
Delivery failed due to the message being deleted.

`#define kTelSmsDSRPermSMNotExist 17`
Delivery failed.

`#define kTelSmsDSRPermOther 18`
Delivery failed due to some other cause.

SMS Report Types

Purpose Report types used in the `reportType` field of the `TelSmsReportType` structure.

Declared In `TelephonyMgr.h`

Constants `#define kTelSmsStatusReportDeliveryType 0`
Status report or delivery acknowledgement.

`#define kTelSmsManualAckDeliveryType 1`
Manual delivery acknowledgement.

Telephony SMS Functions

This section describes the functions used with the SMS service set of the telephony API.

TelSmsDeleteMessage Function

Purpose

Delete an SMS report, delivered message, or submitted message.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelSmsDeleteMessage(UInt16 iRefnum,
   TelAppID iAppId,
   TelSmsDeleteMessageType *ioParamP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioParamP*
A pointer to a `TelSmsDeleteMessageType` structure that specifies the index and type of the message to delete.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
`errNone` upon success or an error code upon failure.

`transId`
The transaction ID of the operation.

`paramP`
Points to the `TelSmsDeleteMessageType` structure passed to this function in the `ioParamP` parameter.

`functionId`
`kTelSmsDeleteMessageMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments If the deleted message has been delivered, the deletion is performed in the current storage, which you can set with the `TelSmsSelectStorage()` function.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSmsSelectStorage()`

TelSmsGetAvailableStorage Function

Purpose Retrieve the list of all available storage on the phone.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSmsGetAvailableStorage(UInt16 iRefnum,
    TelAppID iAppId,
    TelSmsGetAvailableStorageType *ioParamP,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioParamP*
A pointer to a `TelSmsGetAvailableStorageType` structure that is filled in with information about the storage areas available on the phone.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points

to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelSmsGetAvailableStorageType` structure passed to this function in the `ioParamP` parameter.

`functionId`
: `kTelSmsGetAvailableStorageMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments

The count of storage areas available on the phone is stored into the `count` field of the `TelSmsGetAvailableStorageType` structure referenced by `ioParamP`, and the storage ID of each available room is stored into the buffer referenced by the `storagesP` field. If the `storagesP` buffer is too small to contain all of the storage IDs, the buffer is truncated and this function returns the `telErrBufferSize` error. The `count` field of the structure is always updated to contain the total number of available storage areas on the phone.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelSmsSelectStorage()`

TelSmsGetDataMaxSize Function

Purpose

Returns the maximum length, in bytes, of a message on the current network.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelSmsGetDataMaxSize(UInt16 iRefnum,
   TelAppID iAppId, UInt16 *oSizeP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

← *oSizeP*
: A pointer to an unsigned integer value that is updated with the maximum length of an SMS message on the current network.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `oSizeP` parameter.

```
functionId
     kTelSmsGetDataMaxSizeMessage
```

WARNING! When using this function asynchronously, you must ensure that the value referenced by `oSizeP` remains in memory until the asynchronous call completes.

Comments You can use this function to determine the maximum size you need to allocate to read a message.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSmsReadMessage()`, `TelSmsReadMessages()`

TelSmsGetMessageCount Function

Purpose Retrieve the total number of message slots, and the number of filled slots for the specified message type.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSmsGetMessageCount(UInt16 iRefnum,
   TelAppID iAppId,
   TelSmsGetMessageCountType *ioParamP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioParamP*
A pointer to a `TelSmsGetMessageCountType` structure that specifies the message type and is filled in with the count information.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelSmsGetMessageCountType` structure passed to this function in the `ioParamP` parameter.

`functionId`
: `kTelSmsGetMessageCountMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments

The currently selected storage area pertains only to delivered messages; other message types are stored on the phone, but not in a specific storage area. If you specify delivered messages, this function retrieves information about the messages in the currently selected SMS storage area on the phone. If you specify a different message type, this function retrieves information about the messages in the phone.

You specify the message type by assigning one of the SMS Message Type Constants to the `messageType` field of the `TelSmsGetMessageCountType` structure before calling this function.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSmsSelectStorage()`

TelSmsGetSelectedStorage Function

Purpose Retrieve the ID of the currently selected storage area on the phone.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSmsGetSelectedStorage(UInt16 iRefnum,
   TelAppID iAppId, UInt8 *oStorageIdP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

← *oStorageIdP*
A pointer to an unsigned byte value that is assigned the ID of the currently selected storage area on the phone. The assigned ID value is one of the SMS Storage ID Constants.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
`errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `oStorageIdP` parameter.

`functionId`
: `kTelSmsGetSelectedStorageMessage`

WARNING! When using this function asynchronously, you must ensure that the value referenced by `oStorageIdP` remains in memory until the asynchronous call completes.

Comments The currently selected storage area pertains only to delivered messages; other message types are stored on the phone, but not in a specific storage area.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSmsSelectStorage()`

TelSmsGetUniquePartId Function

Purpose Return a unique part identifier to assign to the `partId` field of a submit message.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSmsGetUniquePartId(UInt16 iRefnum,
    TelAppID iAppId, UInt16 *oUniqueIdP,
    UInt16 *ioTransIdP)
```

Parameters

→ *`iRefnum`*
: The telephony manager library reference number.

→ *`iAppId`*
: The telephone application attachment identifier for your application.

← *`oUniqueIdP`*
: A pointer to a unsigned integer value. Upon return, this is the unique part ID value.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `oUniqueIdP` parameter.

`functionId`
: `kTelUrqSmsGetUniquePartIdMessage`

WARNING! When using this function asynchronously, you must ensure that the value referenced by `oUniquePartIdP` remains in memory until the asynchronous call completes.

Comments

This function corresponds to the `kTelUrqSmsGetUniquePartIdMessage` value.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelSmsSendMessage()`

TelSmsReadMessage Function

Purpose Retrieve a delivered message from the currently selected storage area.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSmsReadMessage(UInt16 iRefnum,
    TelAppID iAppId,
    TelSmsDeliveryMessageType *ioMessageP,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioMessageP*
A pointer to a `TelSmsDeliveryMessageType` structure that is filled in with the message.

On input, the `index` field of this structure contains the index of the message that you want retrieved. Message indexes are zero-based.

On input, the `extensionsCount` field specifies the number of extensions allocated in the `extensionsP` array. You must allocate at least one multi-part extension, even for a single-part message.

On input, the `dataSize` field specifies the allocated size of the `data` buffer, and the `originatingAddressSize` field specifies the allocated size of the `originatingAddress` string. Upon return, each size field specifies the complete size of the data that was stored into the buffer, even if the data had to be truncated to fit.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelSmsDeliveryMessageType` structure passed to this function in the `ioMessageP` parameter.

`functionId`
: `kTelSmsReadMessageMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioMessageP` remains in memory until the asynchronous call completes.

Comments

The message data is stored into the `data` field of the `TelSmsDeliveryMessageType` structure referenced by `ioMessageP`. If the complete message data is too large to fit into the allocated size of the `data` field, the message data is truncated and this function returns the `telErrBufferSize` error. The `dataSize` field of the structure is always updated to contain the actual size, in bytes, of the message data.

The originating address string is stored into the `originatingData` field of the `TelSmsDeliveryMessageType` structure referenced by `ioMessageP`. If the complete string is too large to fit into the allocated size of the `originatingData` field, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `originatingAddressSize` field of the structure is always updated to contain the actual size of the string.

Before calling this function, you need to allocate a number of fields and structures in and related to the `TelSmsDeliveryMessage` structure:

- Allocate each address field with a size of at least `kTelMaxPhoneNumberLen + 1`.

  For example, for a GSM message, you must allocate the `originatingAddress` and `serviceCenterNumber` fields in the `TelSmsDeliveryAdvancedGSMType` structure in the `TelSmsDeliveryMessage` structure.
- Allocate the message field data to have the maximum size of a message on the current network. You can determine this value by calling the `TelSmsGetDataMaxSize()` function.
- Allocate at least one `TelSmsExtensionType` structure in the `TelSmsDeliveryMessage` structure. You must have at least one extension structure, even if your message has only a single part. If you do not allocate enough extensions for the message, `TelSmsReadMessage()` returns an error. Palm recommends allocating between 3 and 5 extensions for a message.
- You should not allocate a pointer for user extension data. If you receive user extension data, the user extension pointer will reference a block in the message data. Do not deallocate the user extension data when you release the structure.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSmsSelectStorage()`

TelSmsReadMessages Function

Purpose Retrieve a range of delivered messages from the currently selected storage area.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSmsReadMessages(UInt16 iRefnum,
    TelAppID iAppId,
    TelSmsReadMessagesType *ioParamP,
    UInt16 *ioTransIdP)
```

Parameters → *iRefnum*

The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

↔ *ioParamP*
: A pointer to a `TelSmsReadMessagesType` structure.

On input, the `first` field of this structure specifies the index of the first message to retrieve. Message indexes are zero-based.

On input the `count` field of this structure specifies the allocated size of the `messagesP` buffer. Upon return, the `count` field specifies the actual number of messages that were available, even if that many could not fit into the buffer.

The `messagesP` buffer must contain pointers to `TelSmsDeliveryMessageType` structures that have been allocated. Each of these structures must be initialized as described for the `TelSmsReadMessage()` function.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelSmsReadMessagesType` structure passed to this function in the `ioParamP` parameter.

`functionId`
: `kTelSmsReadMessagesMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` and all of the structures referenced by it remain in memory until the asynchronous call completes.

Comments If the message data or originating address string data for any of the retrieved messages is larger than the allocated size of its corresponding buffer, the data is truncated into the buffer, and this function returns the `telErrBufferSize` error.

For more information about using this function and allocating structures for its use, see the Comments description for the `TelSmsReadMessage()` function.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSmsSelectStorage()`

TelSmsReadReport Function

Purpose Read a report from the currently selected storage area.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSmsReadReport(UInt16 iRefnum,
    TelAppID iAppId, TelSmsReportType *ioReportP,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioReportP*

A pointer to a `TelSmsReportType` structure.

On input the `index` field of this structure contains the index of the message that you want retrieved. Message indexes are zero-based.

On input, the `dataSize` field specifies the allocated size of the `data` buffer, and the `originatingAddressSize` field specifies the allocated size of the `originatingAddress` string. Upon return, each size field specifies the complete size of the data that was stored into the buffer, even the data had to be truncated to fit.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`

`errNone` upon success or an error code upon failure.

`transId`

The transaction ID of the operation.

`paramP`

Points to the `TelSmsReportType` structure passed to this function in the `ioReportP` parameter.

`functionId`

`kTelSmsReadReportMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `iEntryP` remains in memory until the asynchronous call completes.

Comments The report data is stored into the `data` field of the `TelSmsReportType` structure referenced by `ioReportP`. If the complete message data is too large to fit into the allocated size of the `data` field, the report data is truncated and this function returns the `telErrBufferSize` error. The `dataSize` field of the structure is always updated to contain the actual size, in bytes, of the report data.

The originating address string is stored into the `originatingData` field of the `TelSmsReportType` structure referenced by `ioReportP`. If the complete string is too large to fit into the allocated size of the `originatingData` field, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `originatingAddressSize` field of the structure is always updated to contain the actual size of the string.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSmsSelectStorage()`, `TelSmsSendMessage()`

TelSmsReadReports Function

Purpose Retrieve a range of reports from the currently selected storage.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSmsReadReports(UInt16 iRefnum,
    TelAppID iAppId,
    TelSmsReadReportsType *ioParamP,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioParamP*

A pointer to a `TelSmsReadReportsType` structure.

On input, the `first` field of this structure specifies the index of the first report to retrieve. Report indexes are zero-based.

On input the `count` field of this structure specifies the allocated size of the `reportsP` buffer. Upon return, the `count` field specifies the actual number of reports that were available, even if that many could not fit into the buffer.

The `reportsP` buffer must contain pointers to `TelSmsReportType` structures that have been allocated. Each of these structures must be initialized as described for the `TelSmsReadReport()` function.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`

`errNone` upon success or an error code upon failure.

`transId`

The transaction ID of the operation.

`paramP`

Points to the `TelSmsReadReportsType` structure passed to this function in the `ioParamP` parameter.

`functionId`

`kTelSmsReadReportsMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` and all of the structured referenced by it remain in memory until the asynchronous call completes.

Comments If the report data or originating address string data for any of the retrieved messages is larger than the allocated size of its corresponding buffer, the data is truncated into the buffer, and this function returns the `telErrBufferSize` error. For more information, see the Comments description for the `TelSmsReadReport()` function.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSmsSelectStorage()`, `TelSmsSendMessage()`

TelSmsReadSubmittedMessage Function

Purpose Read a previously submitted message that was kept on the phone after being sent.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSmsReadSubmittedMessage(UInt16 iRefnum,
    TelAppID iAppId,
    TelSmsSubmittedMessageType *ioMessageP,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioMessageP*

A pointer to a `TelSmsSubmitMessageType` structure that is filled in with the message.

On input, the `index` field contains the index of the message that you want retrieved. Message indexes are zero-based.

Upon return, the `message` field structure is filled in with the message information. You must initialize the buffer size fields of this structure prior to calling this function, including the `dataSize`, `callbackNumberSize`, `serviceCenterNumberSize`, and `destinationAddressSize` fields. Each of these fields is initialized with the allocated size of its corresponding buffer.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`

`errNone` upon success or an error code upon failure.

`transId`

The transaction ID of the operation.

`paramP`

Points to the `TelSmsSubmitMessageType` structure passed to this function in the `ioMessageP` parameter.

`functionId`

`kTelSmsReadSubmittedMessageMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `iEntryP` remains in memory until the asynchronous call completes.

Comments The message data is stored into the `data` field of the `TelSmsSubmitMessageType` structure referenced by `ioMessageP`. If the complete message data is too large to fit into the allocated size of the `data` field, the end of the message data is truncated and this function returns the `telErrBufferSize` error. The `dataSize` field of the structure is always updated to contain the actual size, in bytes, of the message data.

The same strategy applies to the `callbackNumber` buffer and `callbackNumberSize` fields, the `destinationAddress` buffer and `destinationAddressSize` fields, and the `serviceCenterNumber` buffer and `serviceCenterNumberSize` fields. If the size of the data for any of the buffer fields exceeds the allocated length of the buffer, the end of the data is truncated and this function returns the `telErrBufferSize` error. Each of the size fields is always updated to contain the complete size of the data intended for the buffer.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSmsSelectStorage()`, `TelSmsSendMessage()`

TelSmsReadSubmittedMessages Function

Purpose Retrieve a range of submitted messages from the currently selected storage area.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSmsReadSubmittedMessages(UInt16 iRefnum,
    TelAppID iAppId,
    TelSmsReadSubmittedMessagesType *ioParamP,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioParamP*

A pointer to a `TelSmsReadMessagesType` structure.

On input, the `first` field of this structure specifies the index of the first message to retrieve. Message indexes are zero-based.

On input, the `count` field of this structure specifies the allocated size of the `submittedsP` buffer. Upon return, the `count` field specifies the actual number of messages that were available, even if that many could not fit into the buffer.

The `submittedsP` buffer must contain pointers to `TelSmsSubmittedMessageType` structures that have been allocated. Each of these structures must be initialized as described for the `TelSmsReadSubmittedMessage()` function.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`

`errNone` upon success or an error code upon failure.

`transId`

The transaction ID of the operation.

`paramP`

Points to the `TelSmsReadSubmittedMessagesType` structure passed to this function in the `ioParamP` parameter.

`functionId`

`kTelSmsReadSubmittedMessagesMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` and all of the structured referenced by it remain in memory until the asynchronous call completes.

Comments If the message data or any of the other variable-length data for any of the retrieved messages is larger than the allocated size of its corresponding buffer, the end of the data is truncated, and this function returns the `telErrBufferSize` error. For more information, see the Comments description for the `TelSmsReadSubmittedMessage()` function.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSmsSelectStorage()`, `TelSmsSendMessage()`

TelSmsSelectStorage Function

Purpose Select a storage area on the phone as the current storage area.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSmsSelectStorage(UInt16 iRefnum,
   TelAppID iAppId, UInt8 ioStorageId,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

→ *ioStorageId*
The ID of the storage area. This must be one of the SMS Storage ID Constants.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `ioStorageId` parameter.

`functionId`
: `kTelSmsSelectStorageMessage`

Comments

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelSmsGetAvailableStorage()`, `TelSmsGetSelectedStorage()`

TelSmsSendManualAcknowledge Function

Purpose

Send a manual acknowledgment of a previously received message. Note that this function is not supported on GSM networks.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelSmsSendManualAcknowledge(UInt16 iRefnum,
    TelAppID iAppId, TelSmsManualAckType *ioAckP,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

↔ *ioAckP*
: A pointer to a structure of type `TelSmsManualAckType`. The fields of this structure specify information about the message being acknowledged.

  Upon return, the `messageId` field is filled in with the ID of the acknowledgment.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelSmsManualAckType` structure passed to this function in the `ioAckP` parameter.

```
functionId
     kTelSmsSendManualAcknowledgeMessage
```

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `iEntryP` remains in memory until the asynchronous call completes.

Comments Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

TelSmsSendMessage Function

Purpose Send an SMS message.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelSmsSendMessage(UInt16 iRefnum,
   TelAppID iAppId,
   TelSmsSendMessageType *ioMessageP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioMessageP*
A pointer to a structure of type `TelSmsSendMessageType`.

On input, the `message` field of this structure contains a `TelSmsSubmitMessageType` with the message to send. You must also allocate and zero at least one `TelSmsExtensionType` structure for the multi-part information.

On output, the `messageId` field of this structure is filled in with the ID that was assigned to the sent message.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelSmsSendMessageType` structure passed to this function in the `ioMessageP` parameter.

`functionId`
: `kTelSmsSendMessageMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioMessageP` remains in memory until the asynchronous call completes.

Comments

You need to make multiple calls to the `TelSmsSendMessage()` function to send your message:

- The first call to `TelSmsSendMessage` does not actually send the message. It computes the number of parts and fills in the multi-part extension structures in the `TelSmsSendMessageType` structure. Note that you must allocate at least one extension structure, even for single-part messages.
- Subsequent calls to `TelSmsSendMessage` actually send the data.
- To send an entire message, you need to call `TelSmsSendMessage` in a loop. Terminate the loop when

an error occurs, or when the `byteSend` field of the first `TelSmsExtensionType` structure has the same value as the `dataSize` field of the `TelSmsSendMessageType` structure that represents the message. For example:

```
while (!TelSmsSendMessage(...) &&
        msg.extensionP[0].extension.mp.byteSend != dataSend);
```

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelSmsSelectStorage()`, `TelCfgSetSmsCenter()`

78

Telephony Phone Book

This chapter describes the phone book service set of the telephony API.

For more information about the telephony manager basic services and the different service sets, see Chapter 73, "Telephony Basic Services."

This chapter describes:

- Telephony Phone Book Data Structures
- Telephony Phone Book Constants
- Telephony Phone Book Functions

For more information about using the telephony manager, see Chapter 10, "Telephony Manager," in *Palm OS Programmer's Companion*, vol. II, *Communications*.

Telephony Phone Book Data Structures

This chapter describes the data structures used with the phone book service set of the telephony API.

TelPhbEntryType Struct

Purpose The `TelPhbEntryType` structure describes a single entry in a phone book.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelPhbEntryType {
   UInt16 phoneIndex;
   Char *fullName;
   UInt8 fullNameSize;
   Char *dialNumber;
   UInt8 dialNumberSize;
} TelPhbEntryType
```

Fields

→ `phoneIndex`

The index of the entry in the phone's phone book. This index is always zero-based.

The telephony manager is responsible for converting this index into the physical index in the phone, if required.

← `fullName`

A buffer into which the retrieved full name of the entry is stored.

This string is stored using the local character set of the Palm™ handheld device. The telephony manager is responsible for converting the character set, if required.

Note that if this buffer is too small to contain the entire retrieved string, the string is truncated and the function using this structure generates a `telErrBufferSize` error.

`fullNameSize`

The size of the `fullName string`.

When the structure is used as an input parameter, this is the allocated size, in bytes, of the `fullName` buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the `fullName` buffer is too small to contain all of the retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

← `dialNumber`

A buffer into which the retrieved telephone number of the entry is stored.

Note that if this buffer is too small to contain the entire retrieved string, the string is truncated, and the function using this structure generates a `telErrBufferSize` error.

`dialNumberSize`

The size of the `dialNumber string`.

When the structure is used as an input parameter, this is the allocated size, in bytes, of the `dialNumber` buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the `dialNumber` buffer is too small to contain all of the retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

TelPhbGetAvailablePhonebooksType Struct

Purpose The `TelPhbGetAvailablePhonebooks()` functions uses the `TelPhbGetAvailablePhonebooksType` structure to return a list of the phone books available on the phone.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelPhbGetAvailablePhonebooksType {
   UInt16 count;
   UInt8 *phonebooksP;
} TelPhbGetAvailablePhonebooksType
```

Fields ↔ `count`

The number of entries in the array referenced by `phonebooksP`.

When the structure is used as an input parameter, this is the allocated size, in bytes, of the `phonebooksP` buffer. Upon return, this is the actual size of the retrieved data.

Upon return, this is the actual number of phone book IDs that could be retrieved. If the `phonebooksP` buffer is too small to contain all of the IDs, this field is assigned the actual count, and the function using this structure generates a `telErrBufferSize` error.

← `phonebooksP`
An array of retrieved phone book IDs. Each ID is one of the Phone Book Type Constants.

TelPhbGetEntriesType Struct

Purpose The `TelPhbGetEntries()` function uses the `TelPhbGetEntriesType` structure to return a list of phone entries.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelPhbGetEntriesType {
   UInt16 first;
   UInt16 count;
   TelPhbEntryType *entriesP;
} TelPhbGetEntriesType
```

Fields

→ `first`
The index of the first entry in the array referenced by `entriesP`.

`count`
When the structure is used as an input parameter, this is number of entries that you want retrieved.

Upon return, this is the actual number of entries that were retrieved.

← `entriesP`
An array of pointers to retrieved `TelPhbEntryType` structures.

TelPhbGetEntryCountType Struct

Purpose The `TelPhbGetEntryCount()` function uses the `TelPhbGetEntryCountType` structure to return information about the entries in the currently selected phone book.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelPhbGetEntryCountType {
   UInt16 slots;
   UInt16 count;
} TelPhbGetEntryCountType
```

Fields

← `slots`
The total number of entry slots available in the phone book.

`count`
The number of filled slots in the phone book.

TelPhbGetEntryMaxSizesType Struct

Purpose The `TelPhbGetEntryMaxSizes()` function uses the `TelPhbGetEntryMaxSizesType` structure to return size information about the entries in the currently selected phone book.

Declared In `TelephonyMgrTypes.h`

Prototype

```
typedef struct _TelPhbGetEntryMaxSizesType {
   UInt8 fullNameMaxSize;
   UInt8 dialNumberMaxSize;
} TelPhbGetEntryMaxSizesType
```

Fields

← `fullNameMaxSize`
The largest size of any `fullName` field in the phone book.

← `dialNumberMaxSize`
The largest size of any `dialNumber` field in the phone book.

Telephony Phone Book Constants

This section describes the constants used with the phone book service set of the telephony API.

Phone Book Type Constants

Purpose The phone book type constants specify the type of phone book that is currently selected.

Declared In `TelephonyMgr.h`

Constants

`#define kTelPhbFixedPhonebook 0`
The phone book stored on the phone.

`#define kTelPhbSimPhonebook 1`
The phone book stored on the SIM card.

`#define kTelPhbPhonePhonebook 2`
The phone book stored on the phone.

`#define kTelPhbLastDialedPhonebook 3`
The phone book from which a telephone number was most recently dialed.

`#define kTelPhbSimAndPhonePhonebook 4`
The combined phone and SIM card phone books.

`#define kTelPhbAdaptorPhonebook 5`
The phone book stored on the telephone adaptor.

`#define kTelPhbFirstOemPhonebook 6`
The ID of the first OEM phone book.

You can specify additional OEM phone books by incrementing this value; for example, the second OEM phone book is specified as `kTelPhbFirstOemPhonebook+1`.

Telephony Phone Book Functions

This section describes the functions used with the phone book service set of the telephony API.

TelPhbAddEntry Function

Purpose Add or replace an entry in the currently selected phone book.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelPhbAddEntry(UInt16 iRefnum,
   TelAppID iAppId, TelPhbEntryType *iEntryP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

→ *iEntryP*
A pointer to a `TelPhbEntryType` structure that contains the new entry information.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
`errNone` upon success or an error code upon failure.

`transId`
The transaction ID of the operation.

`paramP`
Points to the `TelPhbEntryType` structure passed to this function in the `iEntryP` parameter.

`functionId`
`kTelPhbAddEntryMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `iEntryP` remains in memory until the asynchronous call completes.

Comments The `phoneIndex` field of the `TelPhbEntryType` structure referenced by `iEntryP` specifies the index at which to write the entry.

Before using this function, you should verify that it is available by calling the `TelIsPhbServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelPhbDeleteEntry()`, `TelPhbSelectPhonebook()`

TelPhbDeleteEntry Function

Purpose Deletes an entry from the currently selected phone book.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelPhbDeleteEntry(UInt16 iRefnum,
    TelAppID iAppId, UInt16 iEntryIndex,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

→ *iEntryIndex*
The zero-based, logical index of the entry that you want deleted. The Telephony Manager computes the physical index.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `iEntryIndex` parameter.

`functionId`
: `kTelPhbDeleteEntryMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `iEntryP` remains in memory until the asynchronous call completes.

Comments Before using this function, you should verify that it is available by calling the `TelIsPhbServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelPhbAddEntry()`, `TelPhbSelectPhonebook()`

TelPhbGetAvailablePhonebooks Function

Purpose Retrieve the list of all phone books available on the phone.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelPhbGetAvailablePhonebooks(UInt16 iRefnum,
    TelAppID iAppId,
    TelPhbGetAvailablePhonebooksType *ioParamP,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

↔ *ioParamP*
: A pointer to a `TelPhbGetAvailablePhonebooksType` structure that lists the available phone books.

  On input, the `count` field of this structure specifies the allocated size of the `phonebookP` buffer. Upon return, the `count` field specifies the actual number of entries retrieved, even if they were truncated to fit into the buffer.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelPhbGetAvailablePhonebooksType` structure passed to this function in the `iEntryP` parameter.

`functionId`
: `kTelPhbGetAvailablePhonebooksMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

Comments

The phone book IDs are stored into the `phonebookP` field of the `TelPhbGetAvailablePhonebooksType` structure referenced by

`ioParamP`. If the `phonebookP` buffer is too small to contain all of the IDs, the information is truncated and this function returns the `telErrBufferSize` error. The `count` field of the structure is always updated to contain the actual number of entries that were retrieved.

Before using this function, you should verify that it is available by calling the `TelIsPhbServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelPhbSelectPhonebook()`

TelPhbGetEntries Function

Purpose Retrieve a range of entries from the currently selected phone book.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelPhbGetEntries(UInt16 iRefnum,
   TelAppID iAppId,
   TelPhbGetEntriesType *ioParamP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioParamP*
A pointer to a `TelPhbGetEntriesType` structure that is updated with the phone book entry information. The first entry retrieved is specified in the `first` field of this structure, which is zero-based; the number of entries retrieved is specified by the `count` field. Thus, the last entry retrieved is specified by:

```
ioParamP->first +
    ioParamP->count-1
```

Upon return, the `count` field of the structure is the number of entries that were actually retrieved.

The `entriesP` field of this structure is a buffer that you allocate to contain the required number of pointers. Each

pointer references a `TelPhbEntryType` structure that you must also preallocate.

On input, the `fullNameSize` and `dialNumberSize` fields of this structure specify the allocated sizes of the `fullName` and `dialNumber` buffers. Upon return, the `fullNameSize` and `dialNumberSize` fields specify the actual sizes of the buffers, even if a string was truncated to fit into the buffer.

↔ *`ioTransIdP`*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelPhbGetEntriesType` structure passed to this function in the `ioEntriesP` parameter.

`functionId`
: `kTelPhbGetEntriesMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` and any structures that it references remain in memory until the asynchronous call completes.

Comments

The phone book information is stored into the `TelPhbEntryType` structures that you preallocate and refer to in the `entriesP` field of the `TelPhbGetEntriesType` referenced by the `ioParamP` parameter.

If any buffer in any of the `TelPhbEntryType` structures is too small, the string intended for that buffer is truncated, and this function returns the `telErrBufferSize` error. In any case, the `fullNameSize` and `dialNumberSize` fields of each `TelPhbEntryType` structure contain the actual size of their respective strings.

If any entries in the specified range are empty, the entry is not retrieved, and the `count` value in the structure is updated.

Before using this function, you should verify that it is available by calling the `TelIsPhbServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelPhbGetEntry()`, `TelPhbSelectPhonebook()`

TelPhbGetEntry Function

Purpose Retrieve one entry from the currently selected phone book.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelPhbGetEntry(UInt16 iRefnum,
   TelAppID iAppId, TelPhbEntryType *ioEntryP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

↔ *ioEntryP*
A pointer to a `TelPhbEntryType` structure that is updated with the phone book entry information.

On input, the `fullNameSize` and `dialNumberSize` fields of this structure specify the allocated sizes of the `fullName` and `dialNumber` buffers. Upon return, the `fullNameSize` and `dialNumberSize` fields specify the actual sizes of the buffers, even if a string was truncated to fit into the buffer.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelPhbEntryType` structure passed to this function in the `ioEntryP` parameter.

`functionId`
: `kTelPhbGetEntryMessage`

Comments

The phone book information is stored into the `TelPhbEntryType` that you preallocate. If either buffer in the structure is too small, the string intended for that buffer is truncated, and this function returns the `telErrBufferSize` error. In any case, the `fullNameSize` and `dialNumberSize` fields of the structure contain the actual size of their respective strings.

Before using this function, you should verify that it is available by calling the `TelIsPhbServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelPhbGetEntries()`, `TelPhbSelectPhonebook()`

TelPhbGetEntryCount Function

Purpose

Retrieve the total number of entries, and the number of filled entries in the currently selected phone book.

Declared In

`TelephonyMgr.h`

Prototype

```
Err TelPhbGetEntryCount(UInt16 iRefnum,
    TelAppID iAppId,
    TelPhbGetEntryCountType *oParamP,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
: The telephony manager library reference number.

→ *iAppId*
: The telephone application attachment identifier for your application.

← *oParamP*
: A pointer to a `TelPhbGetEntryCountType` structure that is updated with information about the number of entries in the phone book.

↔ *ioTransIdP*
: Set the value of this parameter to `NULL` to cause the function to execute synchronously.

  If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelPhbGetEntryCountType` structure passed to this function in the `oParamP` parameter.

```
functionId
     kTelPhbGetEntryCountMessage
```

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `oParamP` remains in memory until the asynchronous call completes.

Comments The total number of slots and the number of filled slots in the currently selected phone book are stored in the `TelPhbGetEntryCountType` structure referenced by `oParamP`.

Before using this function, you should verify that it is available by calling the `TelIsPhbServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelPhbSelectPhonebook()`

TelPhbGetEntryMaxSizes Function

Purpose Retrieves the maximum buffer sizes of any entries in the currently selected phone book.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelPhbGetEntryMaxSizes(UInt16 iRefnum,
   TelAppID iAppId,
   TelPhbGetEntryMaxSizesType *oParamP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

← *oParamP*
A pointer to a `TelPhbGetEntryCountType` structure that is updated with information about the maximum buffer sizes of entries in the phone book.

↔ *ioTransIdP*

Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the `TelPhbGetEntryCountType` structure passed to this function in the `oParamP` parameter.

`functionId`
: `kTelPhbGetEntryMaxSizesMessage`

WARNING! When using this function asynchronously, you must ensure that the structure referenced by `oParamP` remains in memory until the asynchronous call completes.

Comments

The maximum size of any full name entry and the maximum size of any telephone number entry in the currently selected phone book are stored in the `TelPhbGetEntryCountType` structure referenced by `oParamP`.

Before using this function, you should verify that it is available by calling the `TelIsPhbServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelPhbGetEntries()`, `TelPhbGetEntry()`

TelPhbGetSelectedPhonebook Function

Purpose Retrieve the ID of the currently selected phone book.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelPhbGetSelectedPhonebook(UInt16 iRefnum,
   TelAppID iAppId, UInt8 *oPhbIdP,
   UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

← *oPhbIdP*
A pointer to an unsigned byte value. Upon return, this is filled in with the identifier of the currently selected phone book. The identifier is one of the Phone Book Type Constants.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
`errNone` upon success or an error code upon failure.

`transId`
The transaction ID of the operation.

`paramP`
Points to the unsigned integer value passed to this function in the `oPhIdP` parameter.

`functionId`
`kTelPhbGetSelectedPhonebookMessage`

WARNING! When using this function asynchronously, you must ensure that the value referenced by `oPhbIdP` remains in memory until the asynchronous call completes.

Comments Before using this function, you should verify that it is available by calling the `TelIsPhbServiceAvailable()` macro.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `TelPhbSelectPhonebook()`

TelPhbSelectPhonebook Function

Purpose Make the specified phone book the currently selected phone book.

Declared In `TelephonyMgr.h`

Prototype

```
Err TelPhbSelectPhonebook(UInt16 iRefnum,
    TelAppID iAppId, UInt8 iPhbId,
    UInt16 *ioTransIdP)
```

Parameters

→ *iRefnum*
The telephony manager library reference number.

→ *iAppId*
The telephone application attachment identifier for your application.

→ *iPhbId*
The identifier of the phone book that you want selected as the current phone book. This must be one of the Phone Book Type Constants.

↔ *ioTransIdP*
Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

Synchronous Result Returns `errNone` if the function was successful or returns an error code if not successful.

Asynchronous Result

The following fields are updated in the `TelEventType` event that is sent when the operation completes:

`returnCode`
: `errNone` upon success or an error code upon failure.

`transId`
: The transaction ID of the operation.

`paramP`
: Points to the unsigned integer value passed to this function in the `iPhbId` parameter.

`functionId`
: `kTelPhbSelectPhonebookMessage`

Comments

Before using this function, you should verify that it is available by calling the `TelIsPhbServiceAvailable()` macro.

Compatibility

Implemented only if 4.0 New Feature Set is present.

See Also

`TelPhbGetAvailablePhonebooks()`, `TelPhbGetSelectedPhonebook()`

Part IV
Libraries

79

Internet Library

WARNING! The information in this chapter applies only to a very specific set of Palm, Inc. handhelds such as the Palm VII® running Palm OS® version 3.2 or later. You *must* check for the presence of the Wireless Internet Feature Set prior to using the Internet Library. *Never assume that the Internet Library is present on any given class of handheld*.

This chapter provides reference material for the Internet library API:

- Internet Library Data Structures
- Internet Library Constants
- Internet Library Functions

The header file `INetMgr.h` declares the Internet library API. For more information on the Internet library, see the chapter "Network Communication" in the *Palm OS Programmer's Companion*, vol. II, *Communications*.

WARNING! In future OS versions, PalmSource, Inc. does not intend to support or provide backward compatibility for the Internet library API documented in this chapter.

Internet Library Data Structures

INetCompressionTypeEnum Enum

The `INetCompressionTypeEnum` enum indicates the type of compression used for data exchanged via a socket. One of these enumerated types is set as the value of the `inetSockSettingCompressionTypeID` socket setting (a read-only setting).

Prototype

```
typedef enum {
   inetCompressionTypeNone = 0,
   inetCompressionTypeBitPacked,
   inetCompressionTypeLZ77
} INetCompressionTypeEnum;
```

Constants

`inetCompressionTypeNone`
No compression.

`inetCompressionTypeBitPacked`
Custom 5-bit compression scheme. This is typically used for data sent from the Palm Web Clipping Proxy server.

`inetCompressionTypeLZ77`
Not used; reserved for future use.

INetConfigNameType Struct

The `INetConfigNameType` structure holds the name of an Internet library network **configuration**. A configuration is a set of specific values for the Internet library settings. The Internet library defines a set of built-in configuration aliases for common network setups. These aliases point to configurations instead of holding the actual values themselves. You can use an alias anywhere in the API you would use a configuration. System-defined configuration aliases are listed in "Configuration Aliases" on page 1892.

This structure is used in the functions `INetLibConfigIndexFromName()`, `INetLibConfigRename()`, and `INetLibConfigSaveAs()`.

Prototype

```
#define inetConfigNameSize 32;


typedef struct {
   Char name[inetConfigNameSize];
} INetConfigNameType, *INetConfigNamePtr;
```

Fields

`name`
A configuration name (up to 32 characters).

INetContentTypeEnum Enum

The `INetContentTypeEnum` enum specifies the type of content to be exchanged via a socket. One of these enumerated types is set as the value of the `inetSockSettingContentTypeID` socket setting (a read-only setting).

Prototype

```
typedef enum {
   inetContentTypeTextPlain = 0,
   inetContentTypeTextHTML,
   inetContentTypeImageGIF,
   inetContentTypeImageJPEG,
   inetContentTypeApplicationCML,
   inetContentTypeImagePalmOS,
   inetContentTypeOther
} INetContentTypeEnum;
```

Constants

`inetContentTypeTextPlain`
: Not used

`inetContentTypeTextHTML`
: Not used

`inetContentTypeImageGIF`
: Not used

`inetContentTypeImageJPEG`
: Not used

`inetContentTypeApplicationCML`
: Compressed HTML content (format used by the Palm Web Clipping Proxy server and PQAs)

`inetContentTypeImagePalmOS`
: Palm OS® bitmap

`inetContentTypeOther`
: Some undefined content type

INetHTTPAttrEnum Enum

The `INetHTTPAttrEnum` enum specifies HTTP request and response attributes that are set by `INetLibSockHTTPAttrSet()` and returned by `INetLibSockHTTPAttrGet()`.

Prototype

```
typedef enum {
   inetHTTPAttrWhichPart,
   inetHTTPAttrIncHTTP,
   inetHTTPAttrCheckMailHi,
   inetHTTPAttrCheckMailLo,
   inetHTTPAttrReqContentVersion,

  inetHTTPAttrRspSize,
  inetHTTPAttrResult,
   inetHTTPAttrErrDetail,
   inetHTTPAttrReason,

   inetHTTPAttrContentLength,
   inetHTTPAttrContentLengthUncompressed,
   inetHTTPAttrContentLengthUntruncated,
   inetHTTPAttrContentVersion,
   inetHTTPAttrContentCacheID,
   inetHTTPAttrReqSize
} INetHTTPAttrEnum;
```

Constants

`inetHTTPAttrWhichPart`
: An index to the part of the response data desired, if the response data is partitioned into chunks. Write-only.

`inetHTTPAttrIncHTTP`
: A Boolean that, if set, causes HTTP header data to be included as part of the content when retrieving raw data. Applicable only when `inetSettingConvAlgorithm` is set to `ctpConvNone`. Write-only.

`inetHTTPAttrCheckMailHi`
: High-order byte of ID for checking mail. Write-only.

`inetHTTPAttrCheckMailLo`
: Low-order byte of ID for checking mail. Write-only.

`inetHTTPAttrReqContentVersion`
: Desired content version. Represented as 2 low bytes. Lowest byte is minor version, next higher byte is major version. Write-only.

`inetHTTPAttrRspSize`
: Size of entire HTTP (header and data). Read-only.

`inetHTTPAttrResult`
: Transport protocol error code. Read-only.

`inetHTTPAttrErrDetail`
: Server/proxy error code when using the Palm Web Clipping Proxy server. Read-only.

`inetHTTPAttrReason`
: Transport protocol error message. Read-only.

`inetHTTPAttrContentLength`
: Size of response data. Read-only.

`inetHTTPAttrContentLengthUncompressed`
: Size of uncompressed response data. Read-only.

`inetHTTPAttrContentLengthUntruncated`
: Total size of response data (it may have been truncated to less than this). Read-only.

`inetHTTPAttrContentVersion`
: Actual content version. Represented as 2 low bytes. Lowest byte is minor version, next higher byte is major version. Read-only.

`inetHTTPAttrContentCacheID`
: Cache ID for this item. Read-only.

`inetHTTPAttrReqSize`
: Size of request sent. Read-only.

INetSchemeEnum Enum

The `INetSchemeEnum` enum specifies a protocol (http, https, etc.) used by a socket. Specify one of these enumerated types for the `INetSockSettingScheme` socket setting and for the `scheme` parameter to the `INetLibSockOpen()` call.

Prototype

```
typedef enum {
   inetSchemeUnknown = -1,
   inetSchemeDefault = 0,

   inetSchemeHTTP,
   inetSchemeHTTPS,
   inetSchemeFTP,
   inetSchemeGopher,
   inetSchemeFile,
   inetSchemeNews,
```

```
   inetSchemeMailTo,
   inetSchemePalm,
   inetSchemePalmCall,
   inetSchemeMail,
   inetSchemeMac,

   inetSchemeFirst = inetSchemeHTTP,
   inetSchemeLast = inetSchemeMail
} INetSchemeEnum;
```

Constants

`inetSchemeHTTP`
: Use the HTTP protocol.

`inetSchemeHTTPS`
: Use the HTTPS protocol (for a secure connection).

`inetSchemeFTP`
: Use the FTP protocol. Not implemented.

`inetSchemeGopher`
: Use the Gopher protocol. Not implemented.

`inetSchemeFile`
: Launch local PQA file

`inetSchemeNews`
: Use the News protocol. Not implemented.

`inetSchemeMailTo`
: Launch the local messaging application, passing a "to" address.

`inetSchemePalm`
: Launches a local application database. The URL is expected to be in the form cccc.tttt, where cccc is a four character creator name and tttt is a four character database type. This pair of strings is used to identify an application database to receive the launch message via a call to `SysUIAppSwitch()`.

`inetSchemePalmCall`
: Launches a local application database. The URL is expected to be in the form cccc.tttt, where cccc is a four character creator name and tttt is a four character database type. This pair of strings is used to identify an application database to receive the launch message via a call to `SysAppLaunch()`.

`inetSchemeMail`
: Creates a socket for mail I/O.

`inetSchemeMac`
: Handles opening Mac OS file system HTML URLs. For use by the Simulator only.

INetSettingEnum Enum

The `INetSettingEnum` enum specifies a setting to be returned or set by the `INetLibSettingGet()` or `INetLibSettingSet()` calls.

Prototype

```
typedef enum {
   inetSettingCacheSize,
   inetSettingCacheRef,
   inetSettingNetLibConfig,
   inetSettingRadioID,
   inetSettingBaseStationID,
   inetSettingMaxRspSize,
   inetSettingConvAlgorithm,
   inetSettingContentWidth,
   inetSettingContentVersion,
   inetSettingNoPersonalInfo,
   inetSettingUserName,
   inetSettingLast
} INetSettingEnum;
```

Constants

`inetSettingCacheSize`
: Maximum size of cache (in bytes).

`inetSettingCacheRef`
: `DmOpenRef`, reference to cache database. Read-only.

`inetSettingNetLibConfig`
: The index of the net library network configuration to use. This value is saved as part of the preferences for each Internet library configuration. A value of 0 means to use the current configuration.

`inetSettingRadioID`
: 64-bit radio ID. Read-only. Used for wireless connections only.

`inetSettingBaseStationID`
: Radio base station ID. Read-only. Used for wireless connections only.

`inetSettingMaxRspSize`
: Maximum response size, in bytes. The default is 1024 bytes. Write-only.

`inetSettingConvAlgorithm`
: Content conversion desired. Write-only. Possible values include:
`ctpConvCML` (use 5-bit compression scheme),
`ctpConvCML8Bit` (use 5-bit compression scheme, but in 8-bit form for debugging),
`ctpConvCMLLZ77` (use LZ77 compression scheme),
`ctpConvNone` (no conversion; data is returned in native format)

`inetSettingContentWidth`
: Width of the display for content. The default setting is 160 (pixels). Write-only.

`inetSettingContentVersion`
: Content version (encoder version). Write-only. This setting is used to let the server know what encoder version it should use to encode content sent to the Palm client. Normally you don't need to set this value as it is initialized by `INetLibOpen`. The default encoder version is 0x8001.

`inetSettingNoPersonalInfo`
: Send no device ID or zipcode information to the proxy server. This value is saved as part of the preferences for each Internet library configuration.

`inetSettingUserName`
: Not applicable.

INetSockSettingEnum Enum

The `INetSockSettingEnum` enum specifies a socket setting to be returned or set by the `INetLibSockSettingGet()` or `INetLibSockSettingSet()` calls.

Prototype

```
typedef enum {
   inetSockSettingScheme,
   inetSockSettingSockContext,
   inetSockSettingCompressionType,
   inetSockSettingCompressionTypeID,
   inetSockSettingContentType,
   inetSockSettingContentTypeID,
   inetSockSettingData,
   inetSockSettingDataHandle,
   inetSockSettingDataOffset,
   inetSockSettingTitle,
   inetSockSettingURL,
   inetSockSettingIndexURL,
   inetSockSettingFlags,
   inetSockSettingReadTimeout,
   inetSockSettingContentVersion,
   inetSockSettingLast
} INetSockSettingEnum;
```

Constants

`inetSockSettingScheme`
: Requested scheme; one of the `INetSchemeEnum` values. Read-only.

`inetSockSettingSockContext`
: Not used.

`inetSockSettingCompressionType`
: Name of requested compression type. Read-only.

`inetSockSettingCompressionTypeID`
: Requested compression type; one of the `INetCompressionTypeEnum` values. Read-only.

`inetSockSettingContentType`
: String containing the MIME type of the content. Used only on received raw data. Read-only.

`inetSockSettingContentTypeID`
: Content type of socket data; one of the `INetContentTypeEnum` values. Read-only.

`inetSockSettingData`
: Pointer to socket data. Read-only.

`inetSockSettingDataHandle`
: Handle to socket data. Read-only.

`inetSockSettingDataOffset`
: Offset to socket data from handle. Read-only.

`inetSockSettingTitle`
: Web page title. This value is written to the cache (and the Web Clipping Application Viewer uses it later in a history list of cache entries). Write-only.

`inetSockSettingURL`
: URL of requested data. Read-only.

`inetSockSettingIndexURL`
: Index (or master) URL of requested data (for cache indexing). This is the topmost web page in a group of hierarchical pages; it serves to group the pages together and to filter cache list results. The Web Clipping Application Viewer sets this to the URL of the active PQA, for all pages linked from the PQA.

`inetSockSettingFlags`
: URL request flags; one or more of `inetOpenURLFlag`... flags (see URL Open Constants).

`inetSockSettingReadTimeout`
: The default timeout value for reads when the application uses the event mechanism. The time since last receiving data from a socket is monitored and a timeout error status event is returned from `INetLibGetEvent` if the timeout is exceeded.

`inetSockSettingContentVersion`
: Content version number. Read-only.

INetStatusEnum Enum

The `INetStatusEnum` enum specifies the status of the socket. The status is returned in the `inetSockStatusChangeEvent` event structure and by the call `INetLibSockStatus()`.

Prototype

```
typedef enum {
   inetStatusNew,
   inetStatusResolvingName,
   inetStatusNameResolved,
   inetStatusConnecting,
   inetStatusConnected,
```

```
   inetStatusSendingRequest,
   inetStatusWaitingForResponse,
   inetStatusReceivingResponse,
   inetStatusResponseReceived,
   inetStatusClosingConnection,
   inetStatusClosed,
   inetStatusAcquiringNetwork,
   inetStatusPrvInvalid = 30
} INetStatusEnum;
```

Constants

`inetStatusNew`
: Just opened

`inetStatusResolvingName`
: Looking up host address

`inetStatusNameResolved`
: Found host address

`inetStatusConnecting`
: Connecting to host

`inetStatusConnected`
: Connected to host

`inetStatusSendingRequest`
: Sending request

`inetStatusWaitingForResponse`
: Waiting for response

`inetStatusReceivingResponse`
: Receiving response

`inetStatusResponseReceived`
: Response received

`inetStatusClosingConnection`
: Closing connection

`inetStatusClosed`
: Connection closed

`inetStatusAcquiringNetwork`
: Network temporarily unreachable; socket on hold

`inetStatusPrvInvalid`
: Not used

Internet Library Constants

Configuration Aliases

The constants listed here specify Internet library network configuration alias names. Most of the Internet library API requires a configuration index rather than a name. Use `INetLibConfigIndexFromName()` to obtain the alias's index from the name. For more information, see `INetConfigNameType`.

The following aliases are defined for configuration names:

Alias	Name string	Description
`inetCfgNameDefault`	".Default"	Initially points to a generic configuration with no proxy. This uses the configuration set by the user in the Network preferences panel.
`inetCfgNameDefWireline`	".DefWireline"	Initially points to a generic configuration with no proxy. This uses the configuration set by the user in the Network preferences panel.
`inetCfgNameDefWireless`	".DefWireless"	Initially points to a generic configuration with no proxy. This uses the configuration set by the user in the Network preferences panel.
`inetCfgNameCTPDefault`	".CTPDefault"	Initially points to either ".CTPWireless" (on Palm VII® units) or ".CTPWireline" (on all other units). On the Palm VII unit, the Web Clipping Application Viewer application uses this configuration.

Alias	Name string	Description
`inetCfgNameCTPWireline`	".CTPWireline"	Initially points to a wireline configuration that uses the Palm Web Clipping Proxy server.
`inetCfgNameCTPWireless`	".CTPWireless"	Initially points to a wireless configuration that uses the Palm.Net™ wireless system and the Palm Web Clipping Proxy server.

URL Info Constants

The `inetURLInfoFlag`... constants convey information about a URL and are returned by the function `INetLibURLGetInfo()`.

Constant	Value	Description
`inetURLInfoFlagIsSecure`	0x0001	URL was encrypted.
`inetURLInfoFlagIsRemote`	0x0002	URL was retrieved from the net.
`inetURLInfoFlagIsInCache`	0x0004	URL is stored in the cache.

URL Open Constants

The `inetOpenURLFlag`... constants control how the `INetLibURLOpen()` call operates with respect to caching and encryption. These flags are also used for the `inetSockSettingFlags` socket setting.

Constant	Value	Description
`inetOpenURLFlagLookInCache`	0x0001	Read data from the cache, if available.
`inetOpenURLFlagKeepInCache`	0x0002	Store the item in the cache, overwriting any other entries with an equivalent URL.

Constant	Value	Description
`inetOpenURLFlagForceEncOn`	0x0008	Use encryption even if scheme does not desire it.
`inetOpenURLFlagForceEncOff`	0x0010	Do not use encryption even if scheme desires it.

Internet Library Functions

INetLibCacheGetObject Function

Purpose Returns information about an entry in the cache database, including a handle to the record. Either the URL or the unique ID can be used to find the cache entry.

Declared In `INetMgr.h`

Prototype

```
Err INetLibCacheGetObject (UInt16 libRefnum,
   MemHandle clientParamH, UInt8 *urlTextP,
   UInt32 uniqueID, INetCacheInfoPtr cacheInfoP)
```

Parameters

→ *`libRefnum`*
Refnum of the Internet library.

→ *`clientParamH`*
Inet handle allocated by `INetLibOpen`.

→ *`urlTextP`*
Pointer to URL text string to find. If this parameter is `NULL`, then `uniqueID` is used to find the entry.

→ *`uniqueID`*
Unique ID of the cache entry to find. This value can be obtained by calling `INetLibCacheList()`. This parameter is ignored if `urlTextP` is specified.

← *`INetCacheInfoPtr`*
Pointer to a structure where information about the cache entry is returned. See the Comments section for details.

Returns

0	No error
`inetErrParamsInvalid`	One or more of the parameters are invalid.

Comments

The `INetCacheInfoPtr` type returned from this function is defined as a pointer to an `INetCacheInfoType` structure, which has the following definition:

Prototype

```
typedef struct {
   MemHandle recordH;
   INetContentTypeEnum contentType;
   INetCompressionTypeEnum encodingType;
   UInt32 uncompressedDataSize;
   UInt8 flags;
   UInt8 reserved;
   UInt16 dataOffset;
   UInt16 dataLength;
   UInt16 urlOffset;
   UInt32 viewTime;
   UInt32 createTime;
   UInt16 murlOffset;
} INetCacheInfoType, *INetCacheInfoPtr;
```

Fields

`recordH`
: Handle to the cache record.

`contentType`

`encodingType`

`uncompressedDataSize`

`flags`
: Unused.

`reserved`
: Reserved for future use.

`dataOffset`
: Offset to content.

`dataLength`
Size of content.

`urlOffset`
Offset to URL.

`viewTime`
Time last viewed.

`createTime`
Time entry was created.

`murlOffset`
Offset to master URL.

Compatibility Implemented only if Wireless Internet Feature Set is present.

INetLibCacheList Function

Purpose Returns an item from the cache list, based on its URL and index in the list.

Declared In `INetMgr.h`

Prototype

```
Err INetLibCacheList (UInt16 libRefnum,
   MemHandle inetH, UInt8 *cacheIndexURLP,
   UInt16 *indexP, UInt32 *uidP,
   INetCacheEntryP cacheP)
```

Parameters

→ *`libRefnum`*
Refnum of the Internet library.

→ *`inetH`*
Inet handle allocated by `INetLibOpen`.

→ *`cacheIndexURLP`*
Pointer to a master URL string. Cache entries indexed with this master URL are returned. The Web Clipping Application Viewer sets the master URL of a cache page to the URL of the active PQA, so all pages linked from the PQA have the same master URL.

↔ *`indexP`*
Pointer to the index of the entry. Specify an index to find entries at this index or higher in the list. Specify `NULL` to search from the beginning, the first time you call this

function. The index of the entry following the one found is returned on exit.

← *uidP*
Pointer to a long value where the unique ID of the found cache entry is returned.

← *cacheP*
Pointer to a structure where information about the found cache entry is returned. See the Comments section for details.

Returns

0	No error
`inetErrTypeNotCached`	Cache entry under requested index not found
`inetErrParamsInvalid`	The `cacheIndexURLP` parameter is `NULL`
`inetErrCacheInvalid`	The cache database doesn't exist

Comments

This function first sorts the list of cache entries by URL. Then it returns in `uidP` the unique ID of the first cache entry with an index equal to or greater than `indexP`. The `indexP` value is updated to point to the next entry upon return.

To generate a complete list of cache entries having the same master URL (as for a history list), call this function repeatedly, always specifying the updated `index`, until it returns the error `inetErrTypeNotCached`.

Note that a URL can exist multiple times in the Web Clipping Application Viewer cache database, thus the need for the `uidP` value.

The `INetCacheEntryP` type returned from this function is defined as a pointer to an `INetCacheEntryType` structure, which has the following definition:

```
typedef struct {
UInt8 *urlP; // ptr to URL string
UInt16 urlLen; // length of URL string
UInt8 *titleP; // ptr to title string
UInt16 titleLen; // length of title string
UInt32 lastViewed; // time last viewed
                   // seconds since 1/1/1904
```

```
UInt32 firstViewed; // time first viewed
                    // seconds since 1/1/1904
} INetCacheEntryType, *INetCacheEntryP;
```

Compatibility Implemented only if Wireless Internet Feature Set is present.

INetLibCheckAntennaState Function

Purpose Checks the antenna state and displays a dialog asking the user to raise it if it is down.

Declared In `INetMgr.h`

Prototype

```
Err INetLibCheckAntennaState(UInt16 refNum)
```

Parameters

→ *refNum*
Refnum of the Internet library.

Returns

0	The user raised the antenna.
`netErrUserCancel`	The user closed the dialog by tapping Cancel.

This call can also return data manager errors if it fails internally.

Comments Applications don't need to check the antenna state by using this call. If an application opens the Internet library, the Internet library checks the antenna state when needed and displays the dialog to prompt the user to raise the antenna.

Compatibility Implemented only if Wireless Internet Feature Set is present.

INetLibClose Function

Purpose Closes up and frees an inet handle. Closes or decrements the open count of the net library.

Declared In `INetMgr.h`

Prototype

```
Err INetLibClose (UInt16 libRefnum,
   MemHandle inetH)
```

Parameters

→ *libRefnum*
Refnum of the Internet library.

→ *inetH*
Inet handle allocated by `INetLibOpen`.

Returns

0	No error

Comments This call must be made by an application when it's done with the Internet library. It closes any Internet sockets open by the application, disposes the memory referenced by the given inet handle, and calls `NetLibClose`, if necessary, to close the net Library or decrement its open count.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibOpen()

INetLibConfigAliasGet Function

Purpose Determines to which configuration a built-in alias points.

Declared In `INetMgr.h`

Prototype

```
Err INetLibConfigAliasGet (UInt16 refNum,
   UInt16 aliasIndex, UInt16 *indexP,
   Boolean *isAnotherAliasP)
```

Parameters

→ *libRefnum*
Refnum of the Internet library.

→ *aliasIndex*
Index of alias configuration to query. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the

array returned by `INetLibConfigList()`, or the index returned by `INetLibConfigIndexFromName()`.

← *indexP*
Pointer where the index of the configuration pointed to by `aliasIndex` is returned. 0 is returned if `aliasIndex` does not point to another configuration.

← *isAnotherAliasP*
If `*indexP` is the index of another alias configuration, this Boolean is set to `true`.

Returns

0	No error
`inetErrParamsInvalid`	`aliasIndex` is not valid
`inetErrConfigNotAlias`	`aliasIndex` is not an alias configuration

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibConfigAliasSet()

INetLibConfigAliasSet Function

Purpose Points any of the built-in aliases (".DefWireline", ".DefWireless", etc.) to a given defined configuration.

Declared In `INetMgr.h`

Prototype

```
Err INetLibConfigAliasSet (UInt16 refNum,
   UInt16 configIndex, UInt16 aliasToIndex)
```

Parameters

→ *libRefnum*
Refnum of the Internet library.

→ *configIndex*
Index of configuration to set. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by `INetLibConfigList()`, or the index returned by `INetLibConfigIndexFromName()`.

→ *aliasToIndex*
Index of configuration that the alias identified by `configIndex` is to point to. Specify 0 to remove an existing alias assignment.

Returns

0	No error
`inetErrConfigNotAlias`	`configIndex` is not an alias configuration
`inetErrParamsInvalid`	`configIndex` or `aliasToIndex` is not valid
`inetErrConfigCantPointToAlias`	Alias doesn't point to a real entry

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibConfigAliasGet()

INetLibConfigDelete Function

Purpose Deletes a configuration.

Declared In `INetMgr.h`

Prototype

```
Err INetLibConfigDelete (UInt16 refNum,
   UInt16 index)
```

Parameters

→ *refnum*
Refnum of the Internet library.

→ *index*
Index of configuration to delete. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by `INetLibConfigList()`, or the index returned by `INetLibConfigIndexFromName()`.

Returns

0	No error

`inetErrParamsInvalid`	Index not valid
`inetErrConfigCantDelete`	Attempted to delete an alias configuration

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibConfigIndexFromName(), INetLibConfigList()

INetLibConfigIndexFromName Function

Purpose Returns the index of a configuration given it's name.

Declared In `INetMgr.h`

Prototype

```
Err INetLibConfigIndexFromName (UInt16 refNum,
    INetConfigNamePtr nameP, UInt16 *indexP)
```

Parameters

→ *refnum*
Refnum of the Internet library.

→ *nameP*
Pointer to an `INetConfigNameType` structure that names the configuration whose index you want to get.

← *indexP*
Pointer where the index of the configuration identified in `nameP` is returned.

Returns

0	No error
`inetErrConfigNotFound`	Could not find requested configuration name

Comments If you name an alias, this routine returns the index of the alias entry, not the configuration the alias points to. This way the alias can be pointed to a different configuration.

Applications should store the index of the configuration they're using, rather than the name, so that they won't be confused if the user edits the name of the configuration from the Preferences panel.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibConfigList()

INetLibConfigList Function

Purpose Returns an array containing a list of the available Internet library network configurations.

Declared In `INetMgr.h`

Prototype

```
Err INetLibConfigList (UInt16 refNum,
   INetConfigNameType nameArray[],
   UInt16 *arrayEntriesP)
```

Parameters

→ *refnum*
Refnum of the Internet library.

→ *nameArray*
Pointer to an array of `INetConfigNameType` structs that is to be filled in by this routine.

↔ *arrayEntriesP*
On entry, a pointer to the number of entries available in `nameArray`; on exit, a pointer to the total number of entries in the system (which could exceed the size of the array you pass in).

Returns

0	No error

Comments This routine can be used to obtain a list of available configurations for selection by the user.

Note that the built-in alias configurations, which start with a period, should not be displayed to the user as selectable choices. They are designed for internal use by applications that need a predetermined type of service (like ".CTPWireless" for PQA applications.) Also, any configurations that start with an underscore, like "_CTPRAM", should not be displayed. These typically are configurations created by the Internet library for internal use and cannot be edited using the Network preferences panel.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibConfigMakeActive()

INetLibConfigMakeActive Function

Purpose Makes the given configuration active without having to close and reopen the Internet library by using `INetLibOpen`.

Declared In `INetMgr.h`

Prototype

```
Err INetLibConfigMakeActive (UInt16 refNum,
   MemHandle inetH, UInt16 configIndex)
```

Parameters

→ *`libRefnum`*
Refnum of the Internet library.

→ *`inetH`*
Inet handle allocated by `INetLibOpen`.

→ *`configIndex`*
Index of configuration to activate. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by `INetLibConfigList()`, or the index returned by `INetLibConfigIndexFromName()`.

Returns

0	No error

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibConfigSaveAs(), INetLibConfigList(), INetLibConfigIndexFromName()

INetLibConfigRename Function

Purpose Renames a configuration.

Declared In `INetMgr.h`

Prototype

```
Err INetLibConfigRename (UInt16 refNum,
   UInt16 index, INetConfigNamePtr newNameP)
```

Parameters

→ *`libRefnum`*
Refnum of the Internet library.

→ *`index`*
Index of configuration to rename. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the

array returned by `INetLibConfigList()`, or the index returned by `INetLibConfigIndexFromName()`.

→ *newNameP*
Pointer to an `INetConfigNameType` structure holding the new name of the configuration. The name cannot start with a period or an underscore.

Returns

0	No error
`inetErrConfigBadName`	Trying to save as an alias (beginning with ".") or as a built-in configuration (beginning with "_").
`inetErrParamsInvalid`	Invalid index
`inetErrConfigCantDelete`	Configuration to be renamed is either an alias or a built-in configuration

Comments
After renaming, the configuration index stays the same so that applications that are set up to use that configuration will still work correctly. Note that built-in configuration aliases (ones that start with a period) cannot be renamed.

Compatibility
Implemented only if Wireless Internet Feature Set is present.

INetLibConfigSaveAs Function

Purpose
Saves the current network configuration settings under the given name.

Declared In
`INetMgr.h`

Prototype
```
Err INetLibConfigSaveAs (UInt16 refNum,
   MemHandle inetH, INetConfigNamePtr nameP)
```

Parameters
→ *libRefnum*
Refnum of the Internet library.

→ *inetH*
Inet handle allocated by `INetLibOpen`.

→ *nameP*
Pointer to an `INetConfigNameType` structure holding the name of the configuration. The name cannot start with a period or an underscore.

Returns

0	No error
`inetErrConfigBadName`	Trying to save as an alias (beginning with ".") or as a built-in configuration (beginning with "_").
`inetErrConfigTooMany`	The internal configurations table is full. No more entries can be stored.

Comments If the configuration name specified already exists, it is replaced with the new settings.

Compatibility Implemented only if Wireless Internet Feature Set is present.

INetLibGetEvent Function

Purpose A replacement for `EvtGetEvent` that informs an application of status changes to Internet sockets as well as user interface events.

Declared In `INetMgr.h`

Prototype

```
void INetLibGetEvent (UInt16  libRefnum,
   MemHandle inetH, INetEventType *eventP,
   Int32 timeout)
```

Parameters

→ *libRefnum*
Refnum of the Internet library.

→ *inetH*
Inet handle allocated by `INetLibOpen`, or `NULL`.

↔ *eventP*
The event structure is returned via this pointer.

→ *timeout*
Timeout in ticks. Specify `evtWaitForever` to wait forever.

Returns

0	No error

Comments This call is designed to replace `EvtGetEvent()` in applications which use the Internet library. For convenience, if `inetH` is `NULL`, `INetLibGetEvent` is equivalent to `EvtGetEvent`.

`INetLibGetEvent` returns two additional events besides those returned by `EvtGetEvent`: `inetSockReadyEvent` and `inetSockStatusChangeEvent`.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibSockStatus(), INetLibURLOpen(), INetLibSockOpen(), INetLibSockRead()

INetLibOpen Function

Purpose Creates a new application inet handle structure. Opens or increments the open count of the net library.

Declared In `INetMgr.h`

Prototype

```
Err INetLibOpen (UInt16 libRefnum, UInt16 config,
    UInt32 flags, DmOpenRef cacheRef,
    UInt32 cacheSize, MemHandle *inetHP)
```

Parameters

→ *libRefnum*
Refnum of the Internet library. Pass the value "INet.lib" to `SysLibFind()` to return this refnum.

→ *config*
Indicates the type of network service desired by the application. Returned by `INetLibConfigIndexFromName()`.

→ *flags*
Currently unused; set to 0.

→ *cacheRef*
Document cache database reference. Obtain this by using one of the `DmOpenDatabase()`... calls. Pass `NULL` if you don't want to use a cache.

→ *cacheSize*
Maximum size of the document cache (in bytes). This is ignored if you pass `NULL` for `cacheRef`.

← *inetHP*
Pointer to a handle variable.

Returns

0	No error
`inetErrTooManyClients`	Too many clients opened already
`inetErrIncompatibleInterface`	The net library is already open with an incompatible interface

Comments This call must be made by an application before it can use any other Internet library calls. This call opens the Internet library and returns a pointer to an inet handle, which is then passed to subsequent calls to the Internet library. Every application that opens the Internet library gets its own unique inet handle.

When an application is done using the Internet library, it must call `INetLibClose()`, which closes both the Internet library and the net library, if necessary.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibClose(), INetLibConfigIndexFromName()

INetLibSettingGet Function

Purpose Retrieves current settings for an inet handle.

Declared In `INetMgr.h`

Prototype

```
Err INetLibSettingGet (UInt16 libRefnum,
   MemHandle inetH, UInt16 setting, void *bufP,
   UInt16 *bufLenP)
```

Parameters

→ *`libRefnum`*
Refnum of the Internet library.

→ *`inetH`*
Inet handle allocated by `INetLibOpen`.

→ *`setting`*
The setting to get. Specify one of the `INetSettingEnum` enumerated types.

← *`bufP`*
Pointer to buffer where the return value is to be put.

↔ *bufLenP*
Size of `bufP` on entry. Size of setting value on exit.

Returns

0	No error
`inetErrParamsInvalid`	Invalid setting requested
`inetErrSettingSizeInvalid`	`*bufLenP` is the incorrect size for the requested setting

Comments This call can be used to retrieve the current settings of an inet handle. Some settings have default values that are stored in the system preferences database; see `INetSettingEnum` for details.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibOpen(), INetLibSettingSet(), INetSettingEnum

INetLibSettingSet Function

Purpose Changes a setting for an inet handle.

Declared In `INetMgr.h`

Prototype

```
Err INetLibSettingSet (UInt16 libRefnum,
   MemHandle inetH, UInt16 setting, void *bufP,
   UInt16 bufLen)
```

Parameters

→ *libRefnum*
Refnum of the Internet library.

→ *inetH*
Inet handle allocated by `INetLibOpen`.

→ *setting*
The setting to set. Specify one of the `INetSettingEnum` enumerated types.

→ *bufP*
Pointer to the new setting value.

→ *bufLen*
Size of the value in `bufP`.

Returns

0	No error
`inetErrParamsInvalid`	Invalid setting specified
`inetErrSettingSizeInvalid`	`bufLen` is the incorrect size for the specified setting

Comments Any changes made to the settings last only as long as the `inetH` is around (until `INetLibClose()` is called) and do not affect other applications that might be using the Internet library.

An important note is that settings made through this call essentially change the default values for any sockets subsequently created through `INetLibURLOpen()` or `INetLibSockOpen()`.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibSettingGet(), INetSettingEnum

INetLibSockClose Function

Purpose Closes an inet socket handle.

Declared In `INetMgr.h`

Prototype

```
Err INetLibSockClose (UInt16 libRefnum,
   MemHandle socketH)
```

Parameters

→ *`libRefnum`*
Refnum of the Internet library.

→ *`socketH`*
Handle of the socket to close.

Returns

0	No error

Comments This call closes down and releases all memory associated with a socket created by `INetLibSockOpen` or `INetLibURLOpen`.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibOpen(), INetLibSockOpen(), INetLibURLOpen()

INetLibSockConnect Function

Purpose Establishes a connection with a remote host.

Declared In `INetMgr.h`

Prototype

```
Err INetLibSockConnect (UInt16 libRefnum,
   MemHandle sockH, UInt8 *hostnameP,
   UInt16 port,
   Int32 timeou)
```

Parameters

→ *`libRefnum`*
Refnum of the Internet library.

→ *`sockH`*
Handle (allocated by `INetLibSockOpen` or `INetLibURLOpen`) of the socket to connect.

→ *`hostnameP`*
Pointer to host name string; can be dotted decimal text string.

→ *`port`*
Port number, or 0 for default port.

→ *`timeou`*
Timeout in ticks. Specify `evtWaitForever` to wait forever.

Returns

0	No error

Comments This call associates a remote host name and port number with a socket and, depending on the socket protocol, initiates a connection with that remote host.

This call may return immediately before actually finishing the connect. The application can simply go ahead and submit additional calls such as `INetLibSockRead`, or it may wait for the connect to complete by either polling `INetLibSockStatus` until the socket status is `inetStatusConnected` (not recommended), or by waiting for an `inetSockStatusChangeEvent` event from `INetLibGetEvent` and checking the status then (preferred).

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibSockOpen(), INetLibSockStatus(), INetLibGetEvent()

INetLibSockHTTPAttrGet Function

Purpose Queries HTTP request header formed by the local host, or the response header information returned by a remote host.

Declared In `INetMgr.h`

Prototype

```
Err INetLibSockHTTPAttrGet (UInt16 libRefnum,
   MemHandle sockH, UInt16 attr,
   UInt16 attrIndex,
   void *bufP, UInt32 *bufLenP)
```

Parameters

→ *libRefnum*
Refnum of the Internet library.

→ *sockH*
Handle (allocated by `INetLibSockOpen` or `INetLibURLOpen`) of the socket.

→ *attr*
The attribute to get. Specify one of the `INetHTTPAttrEnum` values.

→ *attrIndex*
The attribute index (if any). Currently unused.

← *bufP*
Pointer to the address where the result is returned.

↔ *bufLenP*
Pointer to the size of `bufP` on entry; size of returned value on exit.

Returns

0	No error
`inetErrSettingNotImplemented`	Invalid setting specified
`inetErrSettingSizeInvalid`	`bufLen` is the incorrect size for the specified setting

Comments This call queries either the request header formed by `INetLibSockHTTPReqCreate` and `INetLibSockHTTPAttrSet`, or the response header returned by the remote host.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibSockHTTPReqCreate()

INetLibSockHTTPAttrSet Function

Purpose Adds additional HTTP request headers to an HTTP request in a socket.

Declared In `INetMgr.h`

Prototype

```
Err INetLibSockHTTPAttrSet (UInt16 libRefnum,
   MemHandle sockH, UInt16 attr,
   UInt16 attrIndex,
   UInt8 *bufP, UInt16 bufLen, UInt16 flags)
```

Parameters

→ *libRefnum*
Refnum of the Internet library.

→ *sockH*
Handle (allocated by `INetLibSockOpen` or `INetLibURLOpen`) of the socket.

→ *attr*
The attribute to set. Specify one of the `INetHTTPAttrEnum` values.

→ *attrIndex*
The attribute index (if any). Currently unused.

→ *bufP*
Pointer to additional header text to add.

→ *bufLen*
Length of `bufP`.

→ *flags*
Flags that control the addition of new headers. Currently unused.

Returns

0	No error
`inetErrSettingNotImplemented`	Invalid setting specified
`inetErrSettingSizeInvalid`	`bufLen` is the incorrect size for the specified setting

Comments This call modifies attributes of an HTTP request formed by `INetLibSockHTTPReqCreate`. Generally, attributes are set only before calling `INetLibSockHTTPReqSend`.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibSockHTTPReqCreate(), INetLibSockHTTPReqSend()

INetLibSockHTTPReqCreate Function

Purpose Forms an HTTP request for the socket.

Declared In `INetMgr.h`

Prototype

```
Err INetLibSockHTTPReqCreate (UInt16 libRefnum,
   MemHandle sockH, UInt8 *verbP,
   UInt8 *resNameP,
   UInt8 *refererP)
```

Parameters

→ *libRefnum*
Refnum of the Internet library.

→ *sockH*
Handle (allocated by `INetLibSockOpen` or `INetLibURLOpen`) of the socket.

→ *verbP*
Reserved for future use.

→ *resNameP*
Pointer to a string holding the name of the resource to get or put.

→ *refererP*
Pointer to a string holding the name of the referring URL, or `NULL` if none.

Returns

0	No error
`inetErrParamsInvalid`	Not an HTTP socket

Comments This call forms an HTTP request for the socket. The request is not actually sent to the remote host until `INetLibSockHTTPReqSend()` is called. The HTTP verb used in the request is determined by the value of the `writeP` parameter passed to `INetLibSockHTTPReqSend`: if this parameter is `NULL`, "GET" is used. Otherwise, "POST" is used.

After a call to `INetLibSockHTTPReqCreate` but before a call to `INetLibSockHTTPReqSend`, the application can add additional HTTP request headers using `INetLibSockHTTPAttrSet()`.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibSockHTTPAttrSet(), INetLibSockHTTPReqSend()

INetLibSockHTTPReqSend Function

Purpose Sends an HTTP request to the remote host or looks for data in the cache.

Declared In `INetMgr.h`

Prototype

```
Err INetLibSockHTTPReqSend (UInt16 libRefnum,
   MemHandle sockH, void *writeP,
   UInt32 writeLen,
   Int32 timeout)
```

Parameters

→ *libRefnum*
Refnum of the Internet library.

→ *sockH*
Handle (allocated by `INetLibSockOpen` or `INetLibURLOpen`) of the socket.

→ *writeP*
Pointer to additional data to send after the request headers. Usually used for `POST` and `PUT` operations.

→ *writeLen*
Number of bytes in `writeP`.

→ *timeout*
Timeout in ticks.

Returns

0	No error
`inetErrRequestTooLong`	Request too big
`inetErrEncryptionNotAvail`	Encryption requested but not available

Comments This call sends an HTTP request created by `INetLibSockHTTPReqCreate` and `INetLibSockHTTPAttrSet`

to the remote host. If this is an `POST` or `PUT` operation, the data to write can be specified in `writeP`.

`INetLibSockHTTPReqSend` doesn't always do network I/O. If the proper socket flag is set, it checks first to see if the requested data is already in the cache. If it is, then a pointer to the cached data is stored in the socket and the socket status is updated to show that data is ready to be read. This will trigger an `inetSockReadyEvent` event.

The socket flag (`inetOpenURLFlagLookInCache`) that causes the cache to be checked first can be set via the `flags` parameter to `INetLibURLOpen()` or by calling `INetLibSockSettingSet()` with the appropriate setting (`inetSockSettingFlags`).

After sending the request, the application can wait for the response to arrive by either polling `INetLibSockStatus` until the `inputReady` boolean is set (not recommended), or by waiting for an `inetSockReadyEvent` event from `INetLibGetEvent` (preferred).

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibSockHTTPReqCreate(), INetLibSockHTTPAttrSet(), INetLibGetEvent()

INetLibSockOpen Function

Purpose Creates and returns a new inet socket handle.

Declared In `INetMgr.h`

Prototype

```
Err INetLibSockOpen (UInt16 libRefnum,
   MemHandle inetH, UInt16 scheme,
   MemHandle *sockHP)
```

Parameters

→ *`libRefnum`*
Refnum of the Internet library.

→ *`inetH`*
Inet handle allocated by `INetLibOpen`.

→ *`scheme`*
The protocol scheme to use. Specify one of the `INetSchemeEnum` types.

← *sockHP*
Pointer to the address where the socket handle is returned.

Returns

0	No error
`inetErrTooManySockets`	Too many sockets open
`inetErrSchemeNotSupported`	Requested URL scheme not supported

Comments

This call creates a new socket for the given scheme. No network I/O is performed. This is a relatively low level call that can be used in place of `INetLibURLOpen` when finer control over the socket settings is required.

Using `INetLibURLOpen`, an HTTP request can be handled with the simple sequence: `INetLibURLOpen`, `INetLibSockRead`, and `INetLibSockClose`. When using `INetLibSockOpen`, the same HTTP request would be handled by replacing the `INetLibURLOpen` call with the sequence: `INetLibSockOpen`, `INetLibSockSettingSet` (optional), `INetLibSockConnect`, `INetLibSockHTTPReqCreate`, `INetLibSockHTTPAttrSet` (optional), and `INetLibSockHTTPReqSend`.

The use of `INetLibSockOpen` allows an application finer control over the socket settings (by calling `INetLibSockSettingSet`) and the HTTP request headers (by calling `INetLibSockHTTPAttrSet`).

Compatibility

Implemented only if Wireless Internet Feature Set is present.

See Also

INetLibOpen(), INetLibURLOpen(), INetLibSockRead(), INetLibSockClose(), INetLibSockSettingSet(), INetLibSockHTTPAttrSet()

INetLibSockRead Function

Purpose Reads data from a socket.

Declared In `INetMgr.h`

Prototype

```
Err INetLibSockRead (UInt16 libRefnum,
   MemHandle sockH, void *bufP, UInt32 reqBytes,
   UInt32 *actBytesP, Int32 timeout)
```

Parameters

→ *libRefnum*
Refnum of the Internet library.

→ *sockH*
Inet handle allocated by `INetLibOpen`.

→ *bufP*
Pointer to buffer where the data is placed.

→ *reqBytes*
Requested number of bytes.

← *actBytesP*
Pointer to the actual number of bytes read.

→ *timeout*
Timeout in ticks. Specify `evtWaitForever` to wait forever.

Returns

0	No error

Comments This call attempts to read `reqBytes` bytes from the given socket. It returns the actual number of bytes read in `*actBytesP`. If the connection with the remote host has been closed, `*actBytesP` contains 0 on exit.

Note that it is normal for the actual bytes read to be less than the requested number of bytes. The application should be prepared to call this routine repeatedly until the desired number of bytes have been read or until `*actBytesP` contains 0, indicating the connection has been closed, or until an error is returned.

This call returns as much data as possible without blocking, however, if no data is available to be read, it does block until at least one byte is available.

Normally, applications will wait for an `inetSockReadyEvent` from `INetLibGetEvent()` before calling `INetLibSockRead`.

Alternatively, the application could call `INetLibSockStatus()` to determine if the socket has any data ready (not recommended), or could simply rely on `INetLibSockRead` to block until at least one byte is available to read. If no data is available before the timeout expires, `inetErrReadTimeout` error is returned.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibURLOpen(), INetLibSockOpen(), INetLibSockStatus(), INetLibSockClose(), INetLibGetEvent()

INetLibSockSettingGet Function

Purpose Retrieves a socket setting.

Declared In `INetMgr.h`

Prototype

```
Err INetLibSockSettingGet (UInt16 libRefnum,
   MemHandle socketH, UInt16 setting, void *bufP,
   UInt16 *bufLenP)
```

Parameters

→ *libRefnum*
Refnum of the Internet library.

→ *socketH*
Handle (allocated by `INetLibSockOpen` or `INetLibURLOpen`) of the socket to get a setting from.

→ *setting*
The setting to get. Specify one of the `INetSockSettingEnum` values.

← *bufP*
Pointer to buffer where the setting value is to be placed.

↔ *bufLenP*
Size of `bufP` on entry; size of returned value on exit.

Returns

0	No error
`inetErrParamsInvalid`	Invalid setting requested
`inetErrSettingSizeInvalid`	`*bufLenP` is the incorrect size for the requested setting

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibSockSettingSet()

INetLibSockSettingSet Function

Purpose Changes a setting of a socket.

Declared In `INetMgr.h`

Prototype

```
Err INetLibSockSettingSet (UInt16 libRefnum,
   MemHandle socketH, UInt16 setting, void *bufP,
   UInt16 bufLen)
```

Parameters

→ *`libRefnum`*
Refnum of the Internet library.

→ *`socketH`*
Handle (allocated by `INetLibSockOpen` or `INetLibURLOpen`) of the socket to set.

→ *`setting`*
The setting to set. Specify one of the `INetSockSettingEnum` values.

→ *`bufP`*
Pointer to buffer containing the new setting value.

→ *`bufLen`*
Size of new setting in `bufP`.

Returns

0	No error
`inetErrSettingNotImplemented`	Invalid setting specified
`inetErrSettingSizeInvalid`	`bufLen` is the incorrect size for the setting

Comments This call can be use to override a general setting for a particular socket.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibSockSettingGet()

INetLibSockStatus Function

Purpose Retrieves the current status of a socket.

Declared In `INetMgr.h`

Prototype

```
Err INetLibSockStatus (UInt16 libRefnum,
   MemHandle socketH, UInt16 *statusP,
   Err *sockErrP,
   Boolean *inputReadyP, Boolean *outputReadyP)
```

Parameters

→ *libRefnum*
Refnum of the Internet library.

→ *socketH*
Handle (allocated by `INetLibSockOpen` or `INetLibURLOpen`) of the socket to get status on.

← *statusP*
Pointer to the address where the status is returned. The status will be one of the `INetStatusEnum` values.

← *sockErrP*
Currently unused.

← *inputReadyP*
Pointer to a Boolean; `true` is returned if the socket has data available to read.

← *outputReadyP*
Pointer to a Boolean; `true` is returned if the socket can accept data for writing.

Returns

0	No error

Comments Most applications that use `INetLibGetEvent` will rarely need to use this call since socket status changes are returned in the event structure.

Compatibility Implemented only if Wireless Internet Feature Set is present.

See Also INetLibURLOpen(), INetLibSockOpen(), INetLibSockRead(), INetLibGetEvent()

IINetLibURLCrack Function

Purpose Cracks a URL text string into its components.

Declared In `INetMgr.h`

Prototype

```
Err INetLibURLCrack (UInt16 libRefnum,
   UInt8 *urlTextP, INetURLType *urlP)
```

Parameters

→ *`libRefnum`*
Refnum of the Internet library.

→ *`urlTextP`*
Pointer to URL text string.

↔ *`urlP`*
Pointer to address where the URL information block is to be returned.

Returns

0	No error
`inetErrParamsInvalid`	`urlTextP` is `NULL` or empty, or `urlP` is `NULL`
`inetErrURLVersionInvalid`	`urlP` is wrong version

Comments If a pointer member of `urlP` is set to `NULL` on entry, then on exit it will point to the start of that component within the original `urlTextP` string; the associated member length is set to the length of that URL component. If a pointer member of `urlP` is not `NULL` on entry, then it must point to a buffer of sufficient size to hold the member data, and on exit the component string will be copied into this buffer and the associated member length will be updated with the actual size. Note that the returned strings are not null-terminated, so the length values are important.

It's easiest to initialize the `InetURLType` block to zeros and let this function fill in all the information about the URL components.

The `InetURLType` block returned from this function has the following structure:

```
typedef struct {
UInt16 version; // 0, for future compatibility
UInt8 *schemeP; // ptr to scheme portion
UInt16 schemeLen; // size of scheme portion
UInt16 schemeEnum; // INetSchemeEnum; the scheme
```

```
UInt8 *usernameP; // ptr to username portion
UInt16 usernameLen; // size of username
UInt8 *passwordP; // ptr to password portion
UInt16 passwordLen; // size of password
UInt8 *hostnameP; // ptr to host name portion
UInt16 hostnameLen; // size of host name
UInt16 port; // port number
UInt8 *pathP; // ptr to path portion
UInt16 pathLen; // size of path
UInt8 *paramP; // ptr to param (;param)
UInt16 paramLen; // size of param
UInt8 *queryP; // ptr to query (?query)
UInt16 queryLen; // size of query
UInt8 *fragP; // ptr to fragment (#frag)
UInt16 fragLen; // size of fragment
} INetURLType
```

Compatibility Implemented only if Wireless Internet Feature Set is present.

INetLibURLGetInfo Function

Purpose Returns information about a URL.

Declared In `INetMgr.h`

Prototype

```
Err INetLibURLGetInfo (UInt16 libRefnum,
   MemHandle inetH, UInt8 *urlTextP,
   INetURLInfoType *urlInfoP)
```

Parameters

→ *libRefnum*
Refnum of the Internet library.

→ *inetH*
Inet handle allocated by `INetLibOpen`.

→ *urlTextP*
Pointer to URL text string.

↔ *urlInfoP*
Pointer to address where the URL information structure is to be returned.

Returns

0	No error
`inetErrParamsInvalid`	`urlInfoP` is `NULL` or incorrect version

Comments The `InetURLInfo` block returned from this function has the following structure:

```
typedef struct {
UInt16 version; // 0, for future compatibility
UInt16 flags; // flags word
UInt32 undefined; // reserved for future use
} INetURLInfo
```

The `flags` word can consist of some combination of these values:

```
inetURLInfoFlagIsSecure // URL was encrypted
inetURLInfoFlagIsRemote // URL was retrieved
from the net
inetURLInfoFlagIsInCache // URL is stored in
the cache
```

Compatibility Implemented only if Wireless Internet Feature Set is present.

INetLibURLOpen Function

Purpose Accesses a URL on the Internet or in the cache.

Declared In `INetMgr.h`

Prototype

```
Err INetLibURLOpen (UInt16 libRefnum,
   MemHandle inetH, UInt8 *urlP,
   UInt8 *cacheIndexURLP, MemHandle *sockHP,
   Int32 timeout, UInt16 flags)
```

Parameters

→ *`libRefnum`*
Refnum of the Internet library.

→ *`inetH`*
Inet handle allocated by `INetLibOpen`.

→ *`urlP`*
Pointer to string containing the URL to access.

→ *cacheIndexURLP*
Pointer to URL string under which the requested URL should be indexed in the cache. Specify `NULL` if you don't need to index the cache. If you are using the Web Clipping Application Viewer cache (not recommended), you must follow the Viewer convention, which is to pass the URL of the active PQA.

← *sockHP*
Pointer to address where the socket handle is returned.

→ *timeout*
Timeout in ticks. Specify `evtWaitForever` to wait forever.

→ *flags*
Flags indicating caching and encryption options desired. Specify zero, one, or more of the URL open flags (see URL Open Constants).

Returns

0	No error
`inetErrParamsInvalid`	`urlP` is `NULL`

Comments

This call sets up a connection to a resource on the Internet addressed by `urlP` and returns a socket handle. Note that if you specify that the cache should be searched first, and if the data is found in the cache, no network I/O occurs. The application can then read that socket resource through the `INetLibSockRead()` call.

This call is a convenience routine that internally makes the following calls for http URLs: `INetLibSockOpen`, `INetLibSockConnect`, `INetLibSockHTTPReqCreate`, and `INetLibSockHTTPReqSend`.

This routine returns immediately before performing any required network I/O. It is then up to the caller to either block on `INetLibSockRead`, or to use `INetLibGetEvent()` to model asynchronous operation. Using `INetLibGetEvent` is the preferred way of performing network I/O since it maximizes battery life and user-interface responsiveness.

Compatibility

Implemented only if Wireless Internet Feature Set is present.

See Also

INetLibSockOpen(), INetLibSockConnect(), INetLibSockRead(), INetLibSockClose()

INetLibURLsAdd Function

Purpose Concatenates two URLs, resulting in one absolute URL.

Declared In `INetMgr.h`

Prototype

```
Err INetLibURLsAdd (UInt16 libRefnum,
   Char *baseURLStr, Char *embeddedURLStr,
   Char *resultURLStr, UInt16 *resultLenP)
```

Parameters

→ *libRefnum*
Refnum of the Internet library.

→ *baseURLStr*
Pointer to base URL string.

→ *embeddedURLStr*
Pointer to URL text string to append.

↔ *resultURLStr*
Pointer to resulting URL string.

↔ *resultLenP*
Pointer to size of `resultURLStr` buffer on entry. On exit, pointer to resulting URL length (including null terminator).

Returns

0	No error

Comments Used to append a URL fragment to a base URL, resulting in an absolute URL string that can be passed to `INetLibURLOpen()` or other functions. This routine ensures that the resulting string conforms to the URL format.

Compatibility Implemented only if Wireless Internet Feature Set is present.

INetLibWiCmd Function

Purpose Invokes a command that operates on the wireless indicator.

Declared In `INetMgr.h`

Prototype

```
Boolean INetLibWiCmd (UInt16 refNum, UInt16 cmd,
   int enableOrX, int y)
```

Parameters

→ *refNum*
Refnum of the Internet library.

→ *cmd*
The command to invoke. Specify one of the `WiCmdEnum` values (see Comments section).

→ *enableOrX*
If `cmd` is `wiCmdSetEnabled`, specify 1 to enable the wireless indicator or 0 to disable it. If `cmd` is `wiCmdSetLocation`, this specifies the x coordinate of the location.

→ *y*
The y coordinate of the location. Used only if `cmd` is `wiCmdSetLocation`.

Returns If `cmd` is `wiCmdEnabled`, this function returns `true` if the wireless indicator is enabled or `false` if it is not. For other command types, the return value is undefined.

Comments The wireless indicator is a 19x13 pixel image on the screen to indicate the current wireless signal strength. This shows as 0 - 5 bars. If the application is in a non-modal window with a title bar, the preferred location for the indicator is at (140,1).

It automatically updates itself as long as you are calling `INetLibGetEvent`. It should be shown on screen while a wireless transaction is in progress. It may also be shown when the user has nothing useful to do next but initiate a wireless transaction, and there isn't much other useful information being displayed.

The `WiCmdEnum` enum specifies a command that operates on the wireless indicator in the user interface. The definition of this type is found in `WirelessIndicator.h` and is as follows:

Prototype

```
typedef enum {
   wiCmdInit =0,
   wiCmdClear,
   wiCmdSetEnabled,
   wiCmdDraw,
   wiCmdEnabled,
   wiCmdSetLocation,
   wiCmdErase
} WiCmdEnum;
```

Constants `wiCmdInit`
Initializes the wireless indicator. You must invoke this command first, before using any of the others.

`wiCmdClear`
: Applications shouldn't use this command. To erase the indicator, disable it by using `wiCmdSetEnabled` and passing 0 for `enableOrX`.

`wiCmdSetEnabled`
: Enables or disables the wireless indicator.

`wiCmdDraw`
: Redraws the wireless indicator using the latest data. Applications don't need to use this command since the indicator is redrawn automatically by `INetLibGetEvent`.

`wiCmdEnabled`
: Returns a Boolean indicating if the wireless indicator is enabled.

`wiCmdSetLocation`
: Sets the location for the wireless indicator on the screen.

`wiCmdErase`
: Erases the wireless indicator. Applications shouldn't use this command. To erase the indicator, disable it by using `wiCmdSetEnabled` and passing 0 for `enableOrX`.

Compatibility Implemented only if Wireless Internet Feature Set is present.

80

PalmOSGlue Library

This chapter describes the API provided in the link library PalmOSGlue (`PalmOSGlue.lib` or `libPalmOSGlue.a`).

You use PalmOSGlue if you want to use the international and text manager features described in the chapter "Localized Applications" on page 385 in the *Palm OS Programmer's Companion*, vol. I and you want to maintain backward compatibility with earlier releases of Palm OS®. If you link with PalmOSGlue, include the headers in the folder `Incs\Libraries\PalmOSGlue`, and make calls as they are listed in this chapter, then your code will run regardless of whether the device's version of the operating system implements international support. The code in PalmOSGlue either uses the text manager or international manager on the ROM or, if the managers don't exist, executes a simple Latin equivalent of the function.

NOTE: PalmOSGlue is a link library, not a shared library. Linking with this library increases your application's code size. The amount by which your code size increases varies depending on the number of library functions you call. Use PalmOSGlue only on versions 2.0 and later of Palm OS.

In addition to covering the text and international manager API, PalmOSGlue adds some functions that are not included in any version of the Palm OS. This chapter describes the functions that are unique to PalmOSGlue and provides a mapping of PalmOSGlue calls to calls that exist in later versions of Palm OS.

PalmOSGlue Functions

The following table shows the mapping between the functions declared in the glue headers and the international functions and macros. To learn more about a glue function, click the link in the right column.

This table lists only those functions that map to a function that exists in newer versions of the OS. The functions that are exclusive to PalmOSGlue are not listed. They are described following this table.

Table 80.1 PalmOSGlue function mappings

This PalmOSGlue function...	...is identical to...
BmpGlueGetBitDepth	BmpGetBitDepth()
BmpGlueGetDimensions	BmpGetDimensions()
BmpGlueGetNextBitmap	BmpGetNextBitmap()
DateGlueTemplateToAscii	DateTemplateToAscii()
DateGlueToDOWDMFormat	DateToDOWDMFormat()
FntGlueWCharWidth	FntWCharWidth()
FntGlueWidthToOffset	FntWidthToOffset()
LmGlueGetLocaleSetting	LmGetLocaleSetting()
LmGlueGetNumLocales	LmGetNumLocales()
LmGlueLocaleToIndex	LmLocaleToIndex()
LstGlueGetTopItem	LstGetTopItem()
OmGlueGetCurrentLocale	OmGetCurrentLocale()
OmGlueGetSystemLocale	OmGetSystemLocale()
ResGlueLoadConstant	ResLoadConstant()
SysGlueGetTrapAddress	SysGetTrapAddress()
TsmGlueGetFepMode	TsmGetFepMode
TsmGlueSetFepMode	TsmSetFepMode
TblGlueGetNumberOf Columns	TblGetNumberOfColumns()
TblGlueGetTopRow	TblGetTopRow()
TblGlueSetSelection	TblSetSelection()
TxtGlueByteAttr	TxtByteAttr()

Table 80.1 PalmOSGlue function mappings *(continued)*

This PalmOSGlue function...	...is identical to...
TxtGlueCaselessCompare	TxtCaselessCompare()
TxtGlueCharAttr	TxtCharAttr()
TxtGlueCharBounds	TxtCharBounds()
TxtGlueCharEncoding	TxtCharEncoding()
TxtGlueCharIsAlNum	TxtCharIsAlNum()
TxtGlueCharIsAlpha	TxtCharIsAlpha()
TxtGlueCharIsCntrl	TxtCharIsCntrl()
TxtGlueCharIsDelim	TxtCharIsDelim()
TxtGlueCharIsDigit	TxtCharIsDigit()
TxtGlueCharIsGraph	TxtCharIsGraph()
TxtGlueCharIsHex	TxtCharIsHex()
TxtGlueCharIsLower	TxtCharIsLower()
TxtGlueCharIsPrint	TxtCharIsPrint()
TxtGlueCharIsPunct	TxtCharIsPunct()
TxtGlueCharIsSpace	TxtCharIsSpace()
TxtGlueCharIsUpper	TxtCharIsUpper()
TxtGlueCharIsValid	TxtCharIsValid()
TxtGlueCharSize	TxtCharSize()
TxtGlueCharWidth	FntWCharWidth()
TxtGlueCharXAttr	TxtCharXAttr()
TxtGlueCompare	TxtCompare()
TxtGlueEncodingName	TxtEncodingName()
TxtGlueGetChar	TxtGetChar()

Table 80.1 PalmOSGlue function mappings *(continued)*

This PalmOSGlue function...	...is identical to...
TxtGlueGetNextChar	TxtGetNextChar()
TxtGlueGetPreviousChar	TxtGetPreviousChar()
TxtGlueGetTruncation Offset	TxtGetTruncationOffset()
TxtGlueMaxEncoding	TxtMaxEncoding()
TxtGlueNextCharSize	TxtNextCharSize()
TxtGlueParamString	TxtParamString()
TxtGluePreviousCharSize	TxtPreviousCharSize()
TxtGlueReplaceStr	TxtReplaceStr()
TxtGlueSetNextChar	TxtSetNextChar()
TxtGlueStrEncoding	TxtStrEncoding()
TxtGlueTransliterate	TxtTransliterate()
TxtGlueWordBounds	TxtWordBounds()
WinGlueDrawChar	WinDrawChar()
WinGlueDrawTruncChars	WinDrawTruncChars()

BmpGlueGetBits Function

Purpose Retrieve the bitmap's data. Unlike BmpGetBits(), this function can be used with all versions of Palm OS.

Declared In BmpGlue.h

Prototype

```
void *BmpGlueGetBits (BitmapType *bitmapP)
```

Parameters → *bitmapP*
Pointer to the bitmap's structure (a BitmapType).

Returns Returns a pointer to the bitmap's data.

BmpGlueGetCompressionType Function

Purpose Returns the compression type used for a specified bitmap.

Declared In `BmpGlue.h`

Prototype

```
BitmapCompressionType BmpGlueGetCompressionType
    (const BitmapType *bitmapP)
```

Parameters → *bitmapP*
Pointer to the bitmap.

Returns Returns a `BitmapCompressionType` enum value. If the specified bitmap is not compressed, this function returns `BitmapCompressionTypeNone`. If the specified bitmap's encoding version is less than 2, this function returns `BitmapCompressionTypeScanLine`. If `bitmapP` is `NULL`, this function generates a fatal alert.

See Also `BmpCompress()`

BmpGlueGetTransparentValue Function

Purpose Indicates if a specified bitmap has transparency—if, when the bitmap is drawn, pixels of a certain value won't be drawn.

Declared In `BmpGlue.h`

Prototype

```
Boolean BmpGlueGetTransparentValue
    (const BitmapType *bitmapP,
    UInt32 *transparentValueP)
```

Parameters → *bitmapP*
Pointer to the bitmap.

← *transparentValueP*
The pixel value that isn't drawn, if the bitmap has transparency. If the value returned by `BmpGlueGetTransparentValue` is `false`, `*transparentValueP` is left unchanged.

Returns Returns `true` if, when drawing the specified bitmap, Palm OS doesn't draw pixels that have a value equal to that returned in `transparentValueP`. Returns `false` if all pixels are drawn.

Comments You can specify the transparent color when you create the bitmap using Constructor.

See Also `BmpGlueSetTransparentValue()`

BmpGlueSetTransparentValue Function

Purpose Causes pixels of a specified color not to be drawn when the bitmap is drawn.

Declared In `BmpGlue.h`

Prototype

```
void BmpGlueSetTransparentValue
    (BitmapType *bitmapP, UInt32 transparentValue)
```

Parameters

→ *bitmapP*
Pointer to the bitmap. Note that this must not be a resource bitmap: resource bitmaps cannot be changed.

→ *transparentValueP*
The pixel value that isn't drawn.

Returns Returns nothing.

Comments Does nothing if `bitmapP` is `NULL`, if `bitmapP` is an off-screen bitmap, or if `transparentValue` is invalid given the bitmap's bit depth.

If the specified bitmap's encoding version is less than 2, this function updates it to 2.

See Also `BmpGlueGetTransparentValue()`

CtlGlueGetControlStyle Function

Purpose Return the type of the control, such as button, slider, and so on.

Declared In `CtlGlue.h`

Prototype

```
ControlStyleType CtlGlueGetControlStyle
    (const ControlType *ctlP)
```

Parameters

→ *ctlP*
A pointer to a `ControlType` structure.

Returns Returns one of the `ControlStyleType` constants.

Compatibility Implemented only in the PalmOSGlue library.

CtlGlueGetFont Function

Purpose Gets the font used when drawing a specified control's label.

Declared In `CtlGlue.h`

Prototype

```
FontID CtlGlueGetFont (const ControlType *ctlP)
```

Parameters

→ *ctlP*
Pointer to the control object.

Returns Returns the ID of the font used to draw the control's label.

See Also `CtlGlueSetFont()`

CtlGlueGetGraphics Function

Purpose Gets the bitmaps displayed in place of a specified control's label.

Declared In `CtlGlue.h`

Prototype

```
void CtlGlueGetGraphics (const ControlType *ctlP,
    DmResID *bitmapID, DmResID *selectedBitmapID)
```

Parameters

→ *ctlP*
Pointer to the control.

← *bitmapID*
Resource ID of the bitmap to display when the graphical control is not selected.

← *selectedBitmapID*
Resource ID of the bitmap to display when the graphical control is selected, if, when selected, the graphical control should show a different bitmap.

Returns Returns nothing.

Comments If the specified control is not a graphical control—one that displays a bitmap in place of the text label—`*bitmapID` and `*selectedBitmapID` are set to zero.

This function works with any graphical control, including sliders.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also CtlGlueGetFont(), `CtlSetGraphics()`

CtlGlueIsGraphical Function

Purpose Return whether or not a control is a graphical control.

Declared In `CtlGlue.h`

Prototype

```
Boolean CtlGlueIsGraphical
    (ControlType *controlP)
```

Parameters → *controlP*
Pointer to the control

Returns Returns `true` if the control indicated by `controlP` is a graphical control, a slider, or a feedback slider. Otherwise, it returns `false`.

CtlGlueNewSliderControl Function

Purpose Create a new slider or feedback slider dynamically and install it in the specified form. The newly-created control is marked as a graphical control.

Declared In `CtlGlue.h`

Prototype

```
SliderControlType *CtlGlueNewSliderControl
    (void **formPP, UInt16 ID,
    ControlStyleType style,
    DmResID thumbID, DmResID backgroundID,
    Coord x,
    Coord y, Coord width, Coord height,
    UInt16 minValue, UInt16 maxValue,
    UInt16 pageSize,
    UInt16 value)
```

Parameters ↔ *formPP*
Pointer to the pointer to the form in which the new control is installed. This value is not a handle; that is, the `formPP` value may change if the object moves in memory. In subsequent calls, always use the new `formPP` value returned by this function.

→ *ID*
Symbolic ID of the slider.

→ *style*
Either `sliderCtl` or `feedbackSliderCtl`. See `ControlStyleType`.

→ *thumbID*
Resource ID of the bitmap to display as the slider thumb. The slider thumb is the knob that the user can drag to change the slider's value. To use the default thumb bitmap, pass `NULL` for this parameter.

→ *backgroundID*
Resource ID of the bitmap to display as the slider background. To use the default background bitmap, pass `NULL` for this parameter.

→ *x*
Horizontal coordinate of the upper-left corner of the slider's boundaries, relative to the window in which it appears.

→ *y*
Vertical coordinate of the upper-left corner of the slider's boundaries, relative to the window in which it appears.

→ *width*
Width of the slider, expressed in pixels. Valid values are 1–160.

→ *height*
Height of the slider, expressed in pixels. Valid values are 1–160.

→ *minValue*
Value of the slider when its thumb is all the way to the left.

→ *maxValue*
Value of the slider when its thumb is all the way to the right.

→ *pageSize*
Amount by which to increase or decrease the slider's value when the user clicks to the right or left of the thumb.

→ *value*
The initial value to display in the slider.

Returns Returns a pointer to the new slider control. See `SliderControlType`.

Compatibility Implemented only if 3.5 New Feature Set is present.

See Also `CtlNewSliderControl()`, `CtlNewGraphicControl()`, `CtlNewControl()`, `CtlValidatePointer()`, `FrmRemoveObject()`

CtlGlueSetFont Function

Purpose Sets the font to use when drawing the control's label.

Declared In `CtlGlue.h`

Prototype

```
void CtlGlueSetFont (ControlType *ctlP,
    FontID fontID)
```

Parameters

→ *ctlP*
Pointer to the control object.

→ *fontID*
The ID of the font to use when drawing the control's label.

Returns Returns nothing.

See Also `CtlGlueGetFont()`

CtlGlueSetFrameStyle Function

Purpose Set the frame style for a button control.

Declared In `CtlGlue.h`

Prototype

```
void CtlGlueSetFrameStyle (ControlType *controlP,
    ButtonFrameType frameStyle)
```

Parameters

→ *controlP*
Pointer to a button control.

→ *frameStyle*
Frame style to be applied to the button. Supply one of these values (declared in the `ButtonFrameType` enum): `noButtonFrame`, `standardButtonFrame`, or `boldButtonFrame`. Don't use `rectangleButtonFrame`; rectangular button frames have been deprecated.

Returns Returns nothing.

CtlGlueSetLeftAnchor Function

Purpose Causes a control's left bound to be fixed or to float.

Declared In `CtlGlue.h`

Prototype

```
void CtlGlueSetLeftAnchor (ControlType *ctlP,
    Boolean leftAnchor)
```

Parameters

→ *ctlP*
Pointer to the control.

→ *leftAnchor*
A value of `true` causes the left bound of the control to be fixed.

Returns Returns nothing.

Comments Used by controls that expand and shrink their width when the label is changed.

FldGlueGetLineInfo Function

Purpose Retrieve the word-wrapping information for a visible line within a field.

Declared In `FldGlue.h`

Prototype

```
Boolean FldGlueGetLineInfo
    (const FieldType *fldP,
    UInt16 lineNum, UInt16 *startP,
    UInt16 *lengthP)
```

Parameters

→ *fldP*
A pointer to a `FieldType` structure.

→ *lineNum*
The number of the visible line in the field about which you want to retrieve information. Lines are numbered starting at 0.

← *startP*
The byte offset into the `FieldType`'s `text` field of the first character displayed by this line. If the line is blank, `*startP` is equal to the length of the field's text string.

← *lengthP*
The length in bytes of the portion of the string displayed on this line. If the line is blank, the length is 0.

Returns Returns `true` if `startP` and `endP` contain valid values, or `false` if the field is a single-line field or does not contain a line numbered `lineNum`.

Compatibility Implemented only in the PalmOSGlue library.

FntGlueGetDefaultFontID Function

Purpose Return the font ID of a default font.

Declared In `FntGlue.h`

Prototype

```
FontID FntGlueGetDefaultFontID (
    FontDefaultType inFontType)
```

Parameters → *inFontType*
A `FontDefaultType` constant specifying one of the system default fonts. This value can be one of the following:

`defaultSystemFont`
The default font for the system.

`defaultLargeFont`
The default large font.

`defaultSmallFont`
The default small font.

`defaultBoldFont`
The default bold font.

Returns Returns the ID of `inFontType`.

Comments Use this function whenever you need to obtain a font ID for one of the system default fonts. The default fonts (and thus, the IDs for the default fonts) vary depending on the system's locale. For example, Japanese systems have a different set of default fonts than systems using the Latin character encoding.

Use this function in place of the constants that specify the IDs of default fonts, as shown in the following table.

In place of this...	...use FntGlueGetDefaultFontID with this constant...
`stdFont`	`defaultSystemFont` (best for displaying text) or:
	`defaultSmallFont` (if you want a smaller font)
`largeFont`	`defaultLargeFont`
`largeBoldFont`	`defaultLargeFont`
`boldFont`	`defaultBoldFont`

Note that `defaultSystemFont` and `defaultSmallFont` might return the same font ID or different font IDs, depending on the system locale.

Compatibility Implemented only in the PalmOSGlue library.

See Also `FontSelect()`, `FntGetFont()`, `FntSetFont()`

FrmGlueGetActiveField Function

Purpose Return the active field for a form.

Declared In `FrmGlue.h`

Prototype

```
extern FieldType *FrmGlueGetActiveField
    (const FormType *formP)
```

Parameters

→ *formP*
Pointer to the form, or `NULL` to use the active form.

Returns Returns a pointer to the active field. Returns `NULL` if there is no active form or field.

FrmGlueGetDefaultButtonID Function

Purpose Gets the resource ID of the object on the form defined as the default button.

Declared In `FrmGlue.h`

Prototype

```
UInt16 FrmGlueGetDefaultButtonID
    (const FormType *formP)
```

Parameters → *formP*
Pointer to the form.

Returns Returns the resource ID of the object defined as the default button.

See Also `FrmDoDialog()`, `FrmGlueSetDefaultButtonID()`

FrmGlueGetHelpID Function

Purpose Gets the resource ID number of the form's help resource.

Declared In `FrmGlue.h`

Prototype

```
UInt16 FrmGlueGetHelpID (const FormType *formP)
```

Parameters → *formP*
Pointer to the form.

Returns Returns the resource ID number of the form's help resource. The help resource is a String resource (type `'tSTR'`).

See Also `FrmGlueSetHelpID()`

FrmGlueGetLabelFont Function

Purpose Gets the font used for a particular label that appears on a form.

Declared In `FrmGlue.h`

Prototype

```
FontID FrmGlueGetLabelFont
    (const FormType *formP,
    UInt16 labelID)
```

Parameters → *formP*
Pointer to the form.

→ *labelID*
ID of a label object in the form (the object's type must be `frmLabelObj`).

Returns Returns a `FontID` value of 0 if either `labelID` is invalid or if the object indicated by `labelID` has a type other than `frmLabelObj`. Otherwise, this function returns the ID of the font used for the label.

See Also `FrmGetObjectType()`, `FrmGlueSetLabelFont()`

FrmGlueGetMenuBarID Function

Purpose Gets the ID number of the form's menu bar.

Declared In `FrmGlue.h`

Prototype

```
UInt16 FrmGlueGetMenuBarID
    (const FormType *formP)
```

Parameters → *formP*
Pointer to the form.

Returns Returns the ID number of the form's menu bar, or zero if the form doesn't have a menu bar.

See Also `FrmSetMenu()`

FrmGlueGetObjIDFromObjPtr Function

Purpose Return the ID of a form object given only a pointer to the form object and an indication of the object's type—a pointer to the form isn't required.

Declared In `FrmGlue.h`

Prototype

```
UInt16 FrmGlueGetObjIDFromObjPtr (void *formObjP,
    FormObjectKind objKind)
```

Parameters → *formObjP*
Pointer to a user interface object appearing on a form.

→ *objKind*
The form object's type. Supply one of the values declared in the `FormObjectKind` enum.

Returns Returns the form object's ID, or `frmInvalidObjectID` if `objKind` is not a valid form object type.

See Also `FrmGetObjectId()`, `FrmGetObjectPtr()`, `FrmGetObjectType()`

FrmGlueGetObjectUsable Function

Purpose Returns whether an object in a form has been hidden.

Declared In `FrmGlue.h`

Prototype

```
Boolean
FrmGlueGetObjectUsable (const FormType *formP,
    UInt16 objIndex)
```

Parameters

→ *formP*
A pointer to a `FormType` structure.

→ *objIndex*
The index of the object on the form.

Returns Returns `true` if the object is usable, meaning that it is considered part of the user interface. Returns `false` if the object is not usable. Objects that are not usable never appear on the screen. The function `FrmHideObject()` clears an object's usable bit to hide that the object.

Comments Implemented only in the PalmOSGlue library.

FrmGlueNavDrawFocusRing Function

Purpose Tells the system to draw a focus ring for an object.

Declared In `FrmGlue.h`

Prototype

```
Err FrmGlueNavDrawFocusRing (FormType *formP,
    UInt16 objectID, Int16 extraInfo,
    RectangleType *boundsInsideRingP,
    FrmNavFocusRingStyleEnum ringStyle,
    Boolean forceRestore)
```

Parameters

→ *formP*
Pointer to the form containing the object whose ring is to be drawn.

→ *objectID*
ID of object getting the focus ring.

→ *extraInfo*
Any information that you want associated with the focus ring. For example, the system uses this field to store the temporarily-selected item of a focused, embedded list. Pass `frmNavFocusRingNoExtraInfo` if you have no information to associate with the ring.

→ *boundsInsideRingP*
Pointer to a `RectangleType` structure defining the bounds around which the ring should be drawn.

→ *ringStyle*
One of the `FrmNavFocusRingStyleEnum` values that represents the desired style for the ring. Pass `frmNavFocusRingStyleObjectTypeDefault` if you want the default ring style for the object's type.

→ *forceRestore*
Specifies whether, if the ring is already on the object, what is behind the ring should be restored prior to redrawing the ring.

Returns `errNone` if the focus ring was drawn successfully, or an error code such as one of the following otherwise:

`uilibErrInvalidParam`
formP or *boundsInsideRingP* is `NULL`.

`uilibErrInvalidFocusObject`
objectID specifies a control that is not marked as usable.

`uilibErrFormDoesNotHaveFocus`
The specified form doesn't currently have the focus.

`uilibErrObjectFocusModeOff`
The specified form is not in object focus mode, or the device does not support one-handed navigation.

Comments Note that if one-handed navigation is not supported, this function returns `uilibErrObjectFocusModeOff`. You'll probably want to call `FrmGlueNavIsSupported()` early on in your application,

and then only call the other one-handed navigation functions if `FrmGlueNavIsSupported()` returned `true`.

See Also `FrmGlueNavGetFocusRingInfo()`, `FrmGlueNavObjectTakeFocus()`, `FrmGlueNavRemoveFocusRing()`, `FrmNavDrawFocusRing()`

FrmGlueNavGetFocusRingInfo Function

Purpose If a form has a system-drawn focus ring, gets information about the ring.

Declared In `FrmGlue.h`

Prototype

```
Err FrmGlueNavGetFocusRingInfo
    (const FormType *formP, UInt16 *objectIDP,
    Int16 *extraInfoP,
    RectangleType *boundsInsideRingP,
    FrmNavFocusRingStyleEnum *ringStyleP)
```

Parameters

→ *formP*
Pointer to the form whose ring information is to be obtained.

← *objectIDP*
Supply a pointer to a `UInt16` that, upon exit will contain the ID of the object with the system-drawn focus ring. Pass `NULL` for this parameter if you don't need this information.

← *extraInfoP*
Supply a pointer to a `UInt16` that, upon exit will contain to any extra information associated with the focus ring; pass `NULL` if you don't need this information. "Extra information" is any information that was associated with the ring when `FrmGlueNavDrawFocusRing()` was called. If there is no extra information associated with the focus ring, **extraInfoP* is set to `frmNavFocusRingNoExtraInfo`.

← *boundsInsideRingP*
Supply a pointer to a `RectangleType` structure into which the bounds of the system-drawn focus ring will be written. Pass `NULL` for this parameter if you don't need this information.

← *ringStyleP*
Supply a pointer to a FrmNavFocusRingStyleEnum variable into which the style of the focus ring will be written. Pass NULL for this parameter if you don't need this information.

Returns errNone if the focus ring was drawn successfully, or an error code such as one of the following otherwise:

uilibErrFormDoesNotHaveFocus
The specified form doesn't currently have the focus.

uilibErrObjectFocusModeOff
The specified form is not in object focus mode, or the device does not support one-handed navigation.

uilibErrObjectNotFound
There is no object with a system-drawn focus ring.

Comments For embedded lists in interaction mode, the system stores the temporarily selected item in the extra info field.

Note that if one-handed navigation is not supported, this function returns uilibErrObjectFocusModeOff. You'll probably want to call FrmGlueNavIsSupported() early on in your application, and then only call the other one-handed navigation functions if FrmGlueNavIsSupported() returned true.

See Also FrmGlueNavDrawFocusRing(), FrmGlueNavRemoveFocusRing(), FrmNavGetFocusRingInfo()

FrmGlueNavIsSupported Function

Purpose Determines whether or not the device suppots one-handed navigation.

Declared In FrmGlue.h

Prototype
```
Boolean FrmGlueNavIsSupported (void)
```

Parameters None.

Returns true if one-handed navigation is supported, false otherwise.

Comments This function checks for the presence of either the one-handed functionality defined by PalmSource, or the 5-way navigation

functionality defined by Handspring (palmOne). If this function returns `true`, the `FrmGlueNav...` functions will work on devices of either type.

See Also `FrmGlueNavDrawFocusRing()`, `FrmGlueNavGetFocusRingInfo()`, `FrmGlueNavObjectTakeFocus()`, `FrmGlueNavRemoveFocusRing()`

FrmGlueNavObjectTakeFocus Function

Purpose Sends a `frmObjectFocusTakeEvent` event for an object.

Declared In `FrmGlue.h`

Prototype

```
void FrmGlueNavObjectTakeFocus (FormType *formP,
    UInt16 objID)
```

Parameters

→ *formP*
Pointer to the form that contains the object.

→ *objID*
ID of the object.

Returns Nothing.

Comments If one-handed navigation is not supported on the device this function calls `FrmSetFocus()` for field and table objects and does nothing for all other objects.

Note that if one-handed navigation is not supported, this function returns `uilibErrObjectFocusModeOff`. You'll probably want to call `FrmGlueNavIsSupported()` early on in your application, and then only call the other one-handed navigation functions if `FrmGlueNavIsSupported()` returned `true`.

See Also `FrmNavObjectTakeFocus()`

FrmGlueNavRemoveFocusRing Function

Purpose Removes the system-drawn ring from around the currently-focused object.

Declared In `FrmGlue.h`

Prototype

```
Err FrmGlueNavRemoveFocusRing (FormType *formP)
```

Parameters → *formP*
Pointer to the form containing the object with the focus ring.

Returns `errNone` if the focus ring was drawn successfully, or an error code such as one of the following otherwise:

`uilibErrInvalidParam`
formP is `NULL`.

`uilibErrFormDoesNotHaveFocus`
The specified form doesn't currently have the focus.

`uilibErrObjectFocusModeOff`
The specified form is not in object focus mode, or the device does not support one-handed navigation.

Comments Note that if one-handed navigation is not supported, this function returns `uilibErrObjectFocusModeOff`. You'll probably want to call `FrmGlueNavIsSupported()` early on in your application, and then only call the other one-handed navigation functions if `FrmGlueNavIsSupported()` returned `true`.

See Also `FrmGlueNavDrawFocusRing()`, `FrmGlueNavGetFocusRingInfo()`, `FrmNavRemoveFocusRing()`

FrmGlueSetDefaultButtonID Function

Purpose Designates the object on the form that is to act as the default button.

Declared In `FrmGlue.h`

Prototype

```
void FrmGlueSetDefaultButtonID (FormType *formP,
    UInt16 defaultButton)
```

Parameters → *formP*
Pointer to the form.

→ *defaultButton*
The resource ID of the object on the form that is to be the default button.

Returns Returns nothing.

See Also FrmDoDialog(), FrmGlueGetDefaultButtonID()

FrmGlueSetHelpID Function

Purpose Designates the String resource that is to act as the form's help resource.

Declared In FrmGlue.h

Prototype

```
void FrmGlueSetHelpID (FormType *formP,
   UInt16 helpRscID)
```

Parameters

→ *formP*
Pointer to the form.

→ *helpRscID*
The resource ID of the String resource that is to be the form's help resource.

Returns Returns nothing.

See Also FrmGlueGetHelpID()

FrmGlueSetLabelFont Function

Purpose Sets the font used for a particular label that appears on a form.

Declared In FrmGlue.h

Prototype

```
void FrmGlueSetLabelFont (FormType *formP,
   UInt16 labelID, FontID fontID)
```

Parameters

→ *formP*
Pointer to the form.

→ *labelID*
ID of a label object in the form (the object's type must be `frmLabelObj`).

→ *fontID*
ID of the font to be used for the label.

Returns Returns nothing.

Comments This function does nothing if either `labelID` is invalid or if the object indicated by `labelID` has a type other than `frmLabelObj`.

See Also `FrmGetObjectType()`, `FrmGlueGetLabelFont()`

IntlGlueGetRoutineAddress Function

Purpose Return the address of a Text Manager function or of its PalmOSGlue equivalent.

Declared In `IntlGlue.h`

Prototype

```
void *IntlGlueGetRoutineAddress
    (IntlSelector selector,
    const void *latinSymbol)
```

Parameters

→ *selector*
One of the routine selectors defined in `IntlMgr.h`.

→ *latinSymbol*
The corresponding `TxtLatin...` symbol defined in `IntlGlue.h`.

Returns Returns the address of the native Palm OS function if it is defined. If the function is not defined, it returns the address of the corresponding PalmOSGlue function.

Comments Use `IntlGlueGetRoutineAddress` for performance reasons. You can use the address that it returns to call the function at that address without having to go through the International Manager's trap dispatch table. `IntlGlueGetRoutineAddress` is mostly useful for optimizing the performance of Text Manager functions that are called in a tight loop.

To call `IntlGlueGetRoutineAddress`, you must pass both the international trap for the function and the corresponding symbol. For example, to obtain the address of the `TxtGetNextChar` function or of its PalmOSGlue equivalent, you would make this call:

```
myTxtGetNextChar =
  IntlGlueGetRoutineAddress(intlTxtGetNextChar,
    TxtLatinGetNextChar);
```

The returned address is only valid while your application stays locked in memory. For this reason, you should only use the returned address up to the point where your application terminates. When the application starts up again, you should call `IntlGlueGetRoutineAddress` again.

Compatibility Implemented only in the PalmOSGlue library.

See Also `IntlGetRoutineAddress()`

LmGlueGetSystemLocale Function

Purpose Return the system locale and system character encoding.

Declared In `LmGlue.h`

Prototype

```
CharEncodingType LmGlueGetSystemLocale
    (LmLocaleType *oSystemLocale)
```

Parameters

← *oSystemLocale*
Points to an `LmLocaleType` struct that identifies the system locale.

Returns Returns the system character encoding, unless `oSystemLocale` is null. For details on `CharEncodingType`, see "Character Encoding Constants" on page 825.

Compatibility Implemented only if 5.3SC New Feature Set is present.

See Also `OmGetSystemLocale()`

LstGlueGetDrawFunction Function

Purpose Return a pointer to a list's drawing callback function.

Declared In `LstGlue.h`

Prototype

```
void *LstGlueGetDrawFunction (ListType *listP)
```

Parameters

→ *listP*
Pointer to a list object.

Returns Returns a pointer to the callback function that draws the list's items. If a callback function hasn't been set for the specified list, this function returns `NULL`.

See Also `LstSetDrawFunction()`

LstGlueGetFont Function

Purpose Get the font used to draw a list's text strings.

Declared In `LstGlue.h`

Prototype

```
FontID LstGlueGetFont (const ListType *listP)
```

Parameters → *listP*
Pointer to the list.

Returns Returns the ID of the font used to draw all list text strings.

See Also `LstGlueSetFont()`

LstGlueGetItemsText Function

Purpose Get the text strings that represent the items in a list.

Declared In `LstGlue.h`

Prototype

```
Char **LstGlueGetItemsText
    (const ListType *listP)
```

Parameters → *listP*
Pointer to the list.

Returns Returns a pointer to an array of pointers to the text of the list choices.

See Also `LstGetSelectionText()`, `LstSetListChoices()`

LstGlueSetFont Function

Purpose Specify the font to be used to draw a list's text strings.

Declared In `LstGlue.h`

Prototype

```
void LstGlueSetFont (ListType *listP,
    FontID fontID)
```

Parameters

→ *listP*
Pointer to the list.

→ *fontID*
ID of the font to be used to draw all list text strings.

Returns Returns nothing.

See Also `LstGlueGetFont()`

LstGlueSetIncrementalSearch Function

Purpose Enables or disables incremental search for a sorted popup list.

Declared In `LstGlue.h`

Prototype

```
void LstGlueSetIncrementalSearch
    (ListType *listP,
    Boolean incrementalSearch)
```

Parameters

→ *listP*
Pointer to the list.

→ *incrementalSearch*
Set to `true` to enable incremental search, `false` to disable it.

Returns Returns nothing.

Comments If incremental search is enabled, when the list is displayed the user can navigate the list by entering up to five characters. The list will scroll to present the first list item that matches the entered characters. This feature only works for popup lists, and only works if the list is sorted and the list items are available to the List Manager (that is, you don't pass `NULL` to `LstSetListChoices`).

MemGluePtrNew Function

Purpose Allocate a new nonmovable chunk in the dynamic heap. Unlike `MemPtrNew()`, this function lets you request chunks larger than 64KB.

Declared In `MemGlue.h`

Prototype
```
MemPtr MemGluePtrNew (UInt32 size)
```

Parameters → *size*
The desired size of the chunk.

Returns Returns a pointer to the new chunk, or `NULL` if unsuccessful.

Comments On versions of Palm OS prior to 3.5, this function acts exactly like `MemPtrNew()` and won't allocate chunks larger than 64K.

SysGlueTrapExists Macro

Purpose Indicates if a given trap exists on the current system.

Declared In `SysGlue.h`

Prototype
```
#define SysGlueTrapExists (trapNum)
```

Parameters → *trapNum*
One of the system trap constants.

Returns Returns `true` if the current operating system defines the system trap `trapNum`, or `false` if the trap does not exist on that version of the operating system.

Compatibility Implemented only in the PalmOSGlue library.

TblGlueGetColumnMasked Function

Purpose Determines whether a particular table column is masked.

Declared In `TblGlue.h`

Prototype
```
Boolean TblGlueGetColumnMasked
   (const TableType *tableP, Int16 column)
```

Parameters → *tableP*
Pointer to the table.

→ *column*
Column number (zero-based).

Returns Returns `true` if the column is masked, `false` otherwise.

Comments If a table cell's column is masked and the cell's row is also masked, the table cell is drawn on the screen but is shaded to obscure the information that it contains.

See Also `TblSetColumnMasked()`

TblGlueGetItemPtr Function

Purpose Return the pointer value stored in a table item. Unlike `TblGetItemPtr()`, this function can be used with all versions of Palm OS.

Declared In `TblGlue.h`

Prototype

```
void *TblGlueGetItemPtr (const TableType *tableP,
    Int16 row, Int16 column)
```

Parameters

→ *tableP*
Pointer to a table object.

→ *row*
Row number of the item (zero-based).

→ *column*
Column number of the item (zero-based).

Returns Returns the item's pointer value or `NULL` if the item does not have a pointer value.

Comments Certain types of table items display the value pointed to by the `ptr` field, while other types display the value stored in the `intValue` field (see the `TableItemType` description for details). An application may set the value of the `ptr` field even for items that use the `intValue` field. This function always returns the value of the `ptr` field.

See Also `TblSetItemPtr()`

TxtGlueConvertEncoding Function

Purpose Convert a text buffer from one character encoding to another. Unlike `TxtConvertEncoding()`, this function will operate on many handhelds running Palm OS 3.5.

Declared In `TxtGlue.h`

Prototype

```
Err TxtGlueConvertEncoding
    (Boolean newConversion,
    TxtConvertStateType *ioStateP,
    const Char *srcTextP, UInt16 *ioSrcBytes,
    CharEncodingType srcEncoding, Char *dstTextP,
    UInt16 *ioDstBytes,
    CharEncodingType dstEncoding, const
    Char *substitutionStr, UInt16 substitutionLen)
```

Parameters

→ *newConversion*
: Set to `true` if this function call is starting a new conversion, or `false` if this function call is a continuation of a previous conversion.

↔ *ioStateP*
: If `newConversion` is `false`, this parameter must point to a `TxtConvertStateType` structure containing the same data used for the previous invocation. If `newConversion` is `true` and no subsequent calls are planned, this parameter can be `NULL`.

→ *srcTextP*
: Pointer to the source text buffer. If `newConversion` is `true`, this must point to the start of a text buffer. If `newConversion` is `false`, it may point to a location in the middle of a text buffer. In either case, it must point to an inter-character boundary.

↔ *ioSrcBytes*
: A pointer to the length, in bytes, of the text starting at `srcTextP` that needs to be converted. Upon return, `*ioSrcBytes` contains the number of bytes successfully processed.

→ *srcEncoding*
: The character encoding that the source text uses. This should be one of the Character Encoding Constants.

→ *dstTextP*
: Pointer to the destination text buffer, or `NULL` if you only want to know how large the destination text buffer must be (the minimum size is returned in `*ioDstBytes`). If not `NULL`, `dstTextP` should point to a location where `TxtGlueConvertEncoding` can begin writing.

→ *ioDstBytes*
: A pointer to the length, in bytes, of `dstTextP`. Upon return, `*ioDstBytes` contains the number of bytes required to represent the source text in the new encoding.

→ *dstEncoding*
: The character encoding to which to convert `srcTextP`. This should be one of the constants documented in "Character Encoding Constants" on page 825. Note that the encoding can be modified, giving you greater control over the conversion process: see "Character Encoding Constant Modifiers" on page 1063.

→ *substitutionStr*
: A string to be substituted for any invalid or inconvertible characters that occur in the source text. This string must already be in the destination encoding. If `NULL`, `TxtGlueConvertEncoding` immediately returns if it encounters an invalid character.

→ *substitutionLen*
: The number of bytes in `substitutionStr`, not including the terminating null byte.

Returns Returns `errNone` upon success, or one of the following if an error occurs:

`txtErrConvertOverflow`
: The destination buffer is not large enough to contain the converted text.

`txtErrConvertUnderflow`
: The end of the source buffer contains a partial character.

`txtErrMalformedText`
: An error in the source text encoding has been discovered.

`txtErrNoCharMapping`
: The device does not contain a mapping between the source and destination encodings for at least one of the characters in `srcTextP`, and there is no substitution string.

`txtErrUnknownEncoding`
: One of the specified encodings is unknown or can't be handled, or the function is being called on a handheld running a version of Palm OS prior to 3.5.

`txtErrUnknownEncodingFallbackCopy`
: Either the source or destination encoding is unknown, and the best fit flag was set in the destination encoding. Before returning this error code, `TxtConvertEncoding` copies anything that is 7-bit ASCII from the source text buffer to the destination text buffer.

Comments

This function converts `ioSrcBytes` of text in `srcTextP` from the `srcEncoding` to the `dstEncoding` character encoding and returns the result in `dstTextP`.

`TxtGlueConvertEncoding` can convert between the following "built-in" encodings and the device encoding:

- UTF-8
- UTF-16, UTF-16BE, UTF-16LE, UCS-2
- UTF-32, UTF-32BE, UTF-32LE, UCS-4[1]

It can also support a variety of encodings that are locale dependent. The only locale for which `TxtGlueConvertEncoding` supports additional encodings is simplified Chinese in those versions of Palm OS that support simplified Chinese. The additional source encodings supported for the simplified Chinese locale are:

- GB2312
- GBK
- ISO2022-CN
- Big5
- Big5+HKSCS

1. UTF-32, UTF-32BE, UTF-32LE, and UCS-4 are only built into handhelds running a version of Palm OS that supports simplified Chinese.

- CP1252
- ISO8859-1
- US ASCII (ISO646)
- GSM

The additional destination encodings supported for the simplified Chinese locale are:

- ISO2022-CN
- GSM

If the `charEncodingDstBestFitFlag` is supported (see "Character Encoding Constant Modifiers" on page 1063) but is *not* set in the destination encoding passed to the function, and you specify an unsupported character encoding for either the source or the destination buffer, `TxtGlueConvertEncoding` returns `txtErrUnknownEncoding`.

You can retrieve the device's encoding using the following function:

```
FtrGet(sysFtrCreator, sysFtrNumEncoding,
  &encoding)
```

If you're converting text that was received from the Internet, the encoding's name is passed along with the text data. Use the `TxtNameToEncoding()` function to convert the name to a `CharEncodingType` value.

The `dstTextP` buffer must be large enough to hold the result of converting `srcTextP` to the specified encoding. You can pass `NULL` for the `dstTextP` parameter to determine the required length of the buffer before actually doing the conversion.; the required length is returned in `ioDstBytes`.

If the function encounters an inconvertible character in the source text, it puts `substitutionStr` in the destination buffer in that character's place and continues the conversion. When the conversion is complete, it returns `txtErrNoCharMapping` to indicate that an error occurred (assuming that no other higher-priority error occurred during the conversion). If `substitutionStr` is `NULL`, the function stops the conversion and immediately returns `txtErrNoCharMapping`. `ioSrcBytes` is set to the offset of the inconvertible character, `dstTextP` contains the

converted string up to that point, and `ioDstBytes` contains the size of the converted text. You can examine the character at `ioSrcBytes` and choose to move past it and continue the conversion. Follow the rules for making repeated calls to `TxtGlueConvertEncoding` as described in the next paragraph.

You can make repeated calls to `TxtGlueConvertEncoding` in a loop if you only want to convert part of the input buffer at a time. When you make repeated calls to this function, the first call should use `true` for `newConversion`, and `srcTextP` should point to the start of the text buffer. All subsequent calls should use the following values:

`newConversion`
: `false`.

`ioStateP`
: The same data that was returned by the previous invocation.

`srcTextP`
: The location where this call should begin converting. Typically, this would be the previous `srcTextP` plus the number of bytes returned in `ioSrcBytes`.

  If you are skipping over an inconvertible character, `srcTextP` must point to the character after that location.

`ioSrcBytes`
: The number of bytes that this function call should convert.

`dstTextP`
: A pointer to a location where this function can begin writing the converted string. You might choose to have each function call write to a different destination buffer. To have successive calls write to the same buffer, pass the previous `dstTextP` plus the number of bytes returned in `ioDstBytes` each time.

`ioDstBytes`
: The number of bytes available for output in the `dstTextP` buffer. In other words, the number of bytes remaining.

Encodings Supported by Various ROMs

The locale module is designed to provide support for Unicode, the device encoding, and a set of related or locale-important encodings.

The following tables summarize the set of encodings supported in `TxtGlueConvertEncoding()` by various ROMs.

The encoding returned by the Locale Manager for `lmChoiceInboundDefaultVObjectEncoding` is supported as a source encoding, and the encodings returned for `lmChoicePrimarySMSEncoding`, `lmChoiceSecondarySMSEncoding`, `lmChoicePrimaryEmailEncoding`, `lmChoiceSecondaryEmailEncoding`, and `lmChoiceOutboundVObjectEncoding` are supported as destination encodings. The exception to this rule is `charEncodingAscii`, which gets returned by `LmGlueGetLocaleSetting(lmChoicePrimaryEmailEncoding)` on systems running a version of Palm OS prior to 5.3SC. This encoding—`charEncodingAscii`—isn't supported as a destination encoding on Palm OS prior to 5.3SC.

Table 80.2 Source encodings for Latin ROMs version 4.0, 4.1, 5.0, 5.1, 5.2

charEncodingUCS2	charEncodingUTF8
charEncodingUTF16LE	charEncodingUTF16BE
charEncodingPalmLatin	

Table 80.3 Destination encodings for Latin ROMs version 4.0, 4.1, 5.0, 5.1, 5.2

charEncodingUCS2	charEncodingUTF8
charEncodingUTF16LE	charEncodingUTF16BE
charEncodingPalmLatin	

Table 80.4 Source encodings for Shift-JIS ROMs version 4.0, 4.1, 5.0, 5.1, 5.2

charEncodingUCS2	charEncodingUTF8

Table 80.4 Source encodings for Shift-JIS ROMs version 4.0, 4.1, 5.0, 5.1, 5.2

charEncodingUTF16LE	charEncodingUTF16BE
charEncodingPalmSJIS	

Table 80.5 Destination encodings for Shift-JIS ROMs version 4.0, 4.1, 5.0, 5.1, 5.2

charEncodingUCS2	charEncodingUTF8
charEncodingUTF16LE	charEncodingUTF16BE
charEncodingPalmSJIS	

Table 80.6 Source encodings for GB ROMs version 5.3SC

charEncodingUCS2	charEncodingUTF8
charEncodingUTF16	charEncodingUTF16LE
charEncodingUTF16BE	charEncodingUTF32
charEncodingUTF32BE	charEncodingUTF32LE
charEncodingUCS4	charEncodingPalmGB
charEncodingGB2312	charEncodingGBK
charEncodingISO2022CN	charEncodingBig5
charEncodingBig5_HKSCS	charEncodingAscii
charEncodingCP1252	charEncodingISO8859_1
charEncodingGSM	

Table 80.7 Destination encodings for GB ROMs version 5.3SC

charEncodingUCS2	charEncodingUTF8
charEncodingUTF16	charEncodingUTF16LE

Table 80.7 Destination encodings for GB ROMs version 5.3SC

charEncodingUTF16BE	charEncodingUTF32
charEncodingUTF32BE	charEncodingUTF32LE
charEncodingUCS4	charEncodingPalmGB
charEncodingGB2312	charEncodingGBK
charEncodingISO2022CN	charEncodingAscii
charEncodingISO8859_1	charEncodingGSM

Compatibility

This function works only on Palm OS 3.5 and later.

In versions of Palm OS that do not support simplified Chinese, the focus of `TxtGlueConvertEncoding` is to convert between Unicode-encoded text and the device's character encoding. For this reason, `TxtGlueConvertEncoding` can only handle conversions between the device's encoding and one of UTF-8, UCS-2, UTF-16LE, or UTF-16BE. If you specify any other character encoding for either the source or the destination buffer, the error code `txtErrUnknownEncoding` is returned.

Prior to Palm OS Garnet version 5.3SC, due to a bug in the implementation of the underlying function the destination buffer must be larger than what is needed in order to accommodate the destination text. If the destination buffer is exactly the right size, an error is returned. Accordingly, if you call `TxtGlueConvertEncoding()` and pass `NULL` for *`dstTextP`* in order to determine the needed buffer size, add one to the returned buffer size when you allocate the buffer.

TxtGlueCharIsVirtual Function

Purpose Return whether a character is a virtual character or not.

Declared In `TxtGlue.h`

Prototype

```
Boolean TxtGlueCharIsVirtual(UInt16 inModifiers,
   WChar inChar)
```

Parameters

→ *inModifiers*
The value passed in the `modifiers` field of the `keyDownEvent`.

→ *inChar*
A character.

Returns Returns `true` if the character `inChar` is a virtual character, `false` otherwise.

Comments Virtual characters are nondisplayable characters that trigger special events in the operating system, such as displaying low battery warnings or displaying the keyboard dialog. Virtual characters should never occur in any data and should never appear on the screen.

Starting in Palm OS 3.1, the command modifier bit is always set in the `keyDownEvent` for a virtual character because the range for virtual characters overlaps the range for "real" characters that should appear on the screen. Earlier releases of the operating system did not always set the command modifier for virtual characters.

You can use this function to test whether a character is virtual or not. Pass the `chr` and `modifiers` fields exactly as you received them in the `keyDownEvent`, and this function performs the appropriate check based on the operating system version.

Compatibility Implemented only in the PalmOSGlue library.

TxtGlueFindString Function

Purpose Perform a case-insensitive search for a string in another string.

Declared In `TextMgr.h`

Prototype

```
Boolean TxtGlueFindString
   (const Char *inSourceStr,
   const Char *inTargetStr,
   UInt32 *outPos, UInt16 *outLength)
```

Parameters

→ *inSourceStr*
Pointer to the string to be searched.

→ *inTargetStr*
Prepared version of the string to be found. This string should either be passed directly from the `strToFind` field in the `sysAppLaunchCmdFind` launch code's parameter block or it should be prepared using `TxtGluePrepFindString()`.

← *outPos*
Pointer to the offset of the match in `inSourceStr`.

← *outLength*
Pointer to the length in bytes of the matching text.

Returns Returns `true` if the function finds `inTargetStr` within `inSourceStr`; `false` otherwise.

If found, the values pointed to by the `outPos` and `outLength` parameters are set to the starting offset and the length of the matching text. Unlike `TxtFindString()`, if the target string is not found, the values pointed to by `outPos` and `outLength` are not necessarily set to 0.

The search that `TxtGlueFindString` performs is locale-dependent. On most ROMs with Latin-based encodings, `TxtGlueFindString` returns `true` only if the string is at the beginning of a word. On Shift-JIS encoded ROMs, `TxtGlueFindString` returns `true` if the string is located anywhere in the word.

Comments Use this function instead of `FindStrInStr()` to support the global system find facility. This function contains an extra parameter, `outLength`, to specify the length of the text that matched. Pass this value to `FindSaveMatch()` in the `appCustom` parameter. Then when your application receives

`sysAppLaunchCmdGoTo`, the `matchCustom` field contains the length of the matching text. You use the length of matching text to highlight the match within the selected record.

You must make sure that the parameters `inSourceStr` and `inTargetStr` point to the start of a valid character. That is, they must point to the first byte of a multi-byte character, or they must point to a single-byte character; if they don't, results are unpredictable.

Compatibility Implemented only in the PalmOSGlue library.

See Also `TxtFindString()`, `TxtCaselessCompare()`

TxtGlueGetHorizEllipsisChar Function

Purpose Return the horizontal ellipsis character.

Declared In `TxtGlue.h`

Prototype

```
WChar TxtGlueGetHorizEllipsisChar (void)
```

Parameters `None.`

Returns Returns the character code for horizontal ellipsis.

Comments Versions 3.1 and higher of the Palm OS use different character codes for the horizontal ellipsis character and the numeric space character than earlier versions did. Use `TxtGlueGetHorizEllipsisChar` to return the appropriate code for horizontal ellipsis regardless of which version of Palm OS your application is run on.

Compatibility Implemented only in the PalmOSGlue library.

See Also `ChrHorizEllipsis()`, `TxtGlueGetNumericSpaceChar()`

TxtGlueGetNumericSpaceChar Function

Purpose Return the numeric space character.

Declared In `TxtGlue.h`

Prototype

```
WChar TxtGlueGetNumericSpaceChar (void)
```

Parameters `None.`

Returns Returns the character code for numeric space.

Comments Versions 3.1 and higher of the Palm OS use different character codes for the horizontal ellipsis character and the numeric space character than earlier versions did. Use `TxtGlueGetNumericSpaceChar` to return the appropriate code for numeric space regardless of which version of Palm OS your application is run on.

Compatibility Implemented only in the PalmOSGlue library.

See Also `ChrNumericSpace()`, `TxtGlueGetHorizEllipsisChar()`

TxtGlueLowerChar Function

Purpose Convert a character to lowercase.

Declared In `TxtGlue.h`

Prototype

```
WChar TxtGlueLowerChar (WChar inChar)
```

Parameters → *inChar*
A character.

Returns Returns the character as a lowercase letter.

Comments This function does not handle the case in which the lowercase version of a character is represented by two or more characters. If you need to handle this situation, call the `TxtGlueLowerStr()` function instead of this one.

Compatibility Implemented only in the PalmOSGlue library.

See Also `TxtGlueUpperChar()`, `TxtGlueLowerStr()`, `TxtGlueUpperStr()`, `TxtGlueTransliterate()`, `TxtTransliterate()`, `StrToLower()`

TxtGlueLowerStr Function

Purpose Convert a string to all lowercase letters.

Declared In `TxtGlue.h`

Prototype

```
void TxtGlueLowerStr (Char *ioString,
    UInt16 inMaxLength)
```

Parameters ↔ *ioString*
The string to be converted.

→ *inMaxLength*
The size of the buffer that contains the string, excluding the terminating null character.

Returns Returns in `ioString` the input string with its letters converted to lowercase.

Comments Converting a string from uppercase to lowercase letters or vice versa may change the size of the string. For this reason, you should always check the size of the `ioString` after this call returns.

You must make sure that the parameter `ioString` points to the start of a a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

This function can only handle characters in the ISO Latin 1 character encoding unless the International Feature Set is present.

Compatibility Implemented only in the PalmOSGlue library.

See Also `TxtGlueUpperStr()`, `TxtGlueLowerChar()`, `TxtGlueUpperChar()`, `StrToLower()` `TxtGlueTransliterate`, `TxtTransliterate()`

TxtGluePrepFindString Function

Purpose Set up for `TxtFindString()` or `FindStrInStr()`.

Declared In `TxtGlue.h`

Prototype

```
void TxtGluePrepFindString (const Char *inSource,
    CharPtr outDest, UInt16 inDstSize)
```

Parameters

→ *inSource*
Pointer to the text to be searched for. Must not be `NULL`.

← *outDest*
The same text as in `inSource` but converted to a suitable format for searching. `outDest` must not be the same address as `inSource`.

→ *inDstSize*
The length in bytes of the area pointed to by `outDest`.

Returns Returns in `outDest` an appropriately converted string.

Comments Use this function to normalize the string to search for before using `TxtGlueFindString()`, `TxtFindString()`, or `FindStrInStr()` to perform a search that is internal to your application. If you use any of these three search routines in response to the `sysAppLaunchCmdFind` launch code, the string that the launch code passes in is already properly normalized for the search.

This function normalizes the string to be searched for. The method by which a search string is normalized varies depending on the version of Palm OS and the character encoding supported by the device.

Only `inDstSize` bytes of `inSource` are written to `outDest`. If necessary to prevent overflow of the destination buffer, not all of `inSource` is converted.

You must make sure that the parameter `inSource` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

Compatibility Implemented only in the PalmOSGlue library.

TxtGlueStripSpaces Function

Purpose Strip trailing and/or leading spaces from a string.

Declared In `TxtGlue.h`

Prototype

```
Char *TxtGlueStripSpaces (Char *ioStr,
   Boolean leading, Boolean trailing)
```

Parameters

↔ *ioStr*
: Any string.

→ *leading*
: If `true`, strip the leading spaces from the string.

→ *trailing*
: If `true`, strip the trailing spaces from the string.

Returns Returns `ioStr` with the specified spaces stripped from it. Note that this function both changes the `ioStr` buffer parameter and returns a pointer to it.

Comments You must make sure that the parameter `ioStr` points to the start of a a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

Compatibility Implemented only in the PalmOSGlue library.

TxtGlueTruncateString Function

Purpose Determine if a string can be displayed in a given number of pixels. If not, truncate the string.

Declared In `TxtGlue.h`

Prototype

```
Boolean TxtGlueTruncateString (Char *ioStringP,
   UInt16 inMaxWidth)
```

Parameters

↔ *`ioStringP`*
: A null-terminated string. Upon return, the string is truncated if necessary so that it can be displayed in `inMaxWidth` pixels.

→ *`inMaxWidth`*
: The maximum width in pixels.

Returns Returns `true` if the string was truncated, or `false` if the string can fit without truncation.

Comments This function determines whether `ioStringP` can be displayed in the specified width without being truncated. If it can, `TxtGlueTruncateString` returns `false`. If the string must be truncated, this function truncates the string to one less than the number of characters that can fit in `inMaxWidth` and then appends an ellipsis (...) character. (If the boundary characters are narrower than the ellipsis, more than one character may be dropped to make room). If `inMaxWidth` is narrower than the width of an ellipsis, the string is set to the empty string.

NOTE: Because this function attempts to null-terminate the supplied string, it should not be used with resources in the storage heap.

Compatibility Implemented only in the PalmOSGlue library.

See Also FntWidthToOffset(), WinDrawTruncChars(), TxtGetTruncationOffset()

TxtGlueUpperChar Function

Purpose Convert a character to uppercase.

Declared In TxtGlue.h

Prototype

```
WChar TxtGlueUpperChar (WChar inChar)
```

Parameters

→ *inChar*
Any character.

Returns Returns the character as an uppercase letter.

Comments This function does not handle the case in which the uppercase version of a character is represented by two or more characters. If you need to handle this situation, call the TxtGlueUpperStr() function instead of this one.

Compatibility Implemented only in the PalmOSGlue library.

See Also TxtGlueLowerChar(), TxtGlueUpperStr() TxtGlueLowerStr(), TxtGlueTransliterate, TxtTransliterate() StrToLower()

TxtGlueUpperStr Function

Purpose Convert a string to all uppercase letters.

Declared In TxtGlue.h

Prototype

```
void TxtGlueUpperStr (Char *ioString,
    UInt16 inMaxLength)
```

Parameters

↔ *ioString*
The string to be converted.

→ *inMaxLength*
The size of the buffer that contains the string, excluding the terminating null character.

Returns Returns in ioString the input string with its letters converted to uppercase.

Comments Converting a string from uppercase to lowercase letters or vice versa may change the size of the string. For this reason, you should always check the size of the `ioString` after this call returns.

You must make sure that the parameter `ioString` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

This function can only handle characters in the ISO Latin 1 character encoding unless the International Feature Set is present.

Compatibility Implemented only in the PalmOSGlue library.

See Also `TxtGlueLowerStr()`, `TxtGlueUpperChar()`, `TxtGlueLowerChar()`, `TxtGlueTransliterate`, `TxtTransliterate()` `StrToLower()`

UIColorGlueGetNumTableEntries Function

Purpose Get the number of entries in the UI color table.

Declared In `UIColorGlue.h`

Prototype

```
UInt16 UIColorGlueGetNumTableEntries (void)
```

Parameters None.

Returns Returns the number of color table entries. Note that this is one greater than the maximum value that can be passed to `UIColorGetTableEntryIndex()`, `UIColorGetTableEntryRGB()`, and `UIColorSetTableEntry()`.

WinGlueGetFrameType Function

Purpose Gets the frame type for a specified window.

Declared In `WinGlue.h`

Prototype

```
FrameType WinGlueGetFrameType
    (const WinHandle winH)
```

Parameters → *winH*
The window's handle.

Returns Returns a `FrameType` value indicating the window's frame style.

See Also `WinGlueSetFrameType()`

WinGlueSetFrameType Function

Purpose Sets the type of frame to be used for a specified window.

Declared In `WinGlue.h`

Prototype

```
void WinGlueSetFrameType (WinHandle winH,
    FrameType frame)
```

Parameters

→ *winH*
The window's handle.

→ *frame*
The style of frame to be used.

Returns Returns nothing.

See Also `WinGlueGetFrameType()`

81

Bluetooth Library: General Functions

The Bluetooth library is a shared library that provides a direct interface to the Bluetooth communication capability of the Palm OS. This chapter presents reference material for the security and utility functions of the Bluetooth library API:

`Security Functions`
: Describes security functions that allow the application to manage a database of trusted devices. Trusted devices don't need to undergo authentication when they reconnect with the local device.

`Utility Functions`
: Describes utility functions and macros that are useful when working with the Bluetooth library. They perform such tasks as converting between host and network byte ordering, and converting between device addresses and strings.

The header file `BtLib.h` declares the Bluetooth library functions and macros. The header file `BtLibTypes.h` declares the data structures that you use with those functions and macros. For more information about using the Bluetooth library, see the *Palm OS Programmer's Companion Supplement: Bluetooth*.

Security Functions

The Bluetooth security functions allow the application to manage a database of devices that have been bonded or paired. To understand this database, it is important to understand the difference between pairing and bonding.

Typically, devices must authenticate with each other every time they connect to each other. This process is called **pairing**. However, the devices can **bond** instead. Bonding is similar to pairing except that both devices remember the link key for the connection. If the two

devices ever want to connect to each other again, they don't need to repeat the pairing process.

Information about paired and bonded devices is stored in a device database on the local device. If a remote device has bonded with the local device, its record remains in the database until it is explicitly deleted. If the remote device has paired with the local device but not bonded with it, the record is removed when the connection to the remote device terminates.

Remote devices that have bonded with the local device are also called **trusted devices**.

BtLibSecurityFindTrustedDeviceRecord Function

Purpose Search the device database for the device with the specified Bluetooth address. Return the index of the corresponding device record in the database.

Declared In `BtLib.h`

Prototype

```
Err BtLibSecurityFindTrustedDeviceRecord
    (UInt16 btLibRefNum,
    BtLibDeviceAddressTypePtr addrP,
    UInt16 *index)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *addrP*
Bluetooth address of remote device.

← *index*
Index of the record.

Returns Returns `btLibErrNoError` if successful. Returns `btLibErrNotFound` if a record with the specified remote device address could not be found.

See Also `BtLibSecurityRemoveTrustedDevice Record()`, `BtLibSecurityGetTrustedDeviceRecordInfo()`

BtLibSecurityGetTrustedDeviceRecordInfo Function

Purpose Get information from a device record in the device database.

Declared In `BtLib.h`

Prototype

```
Err BtLibSecurityGetTrustedDeviceRecordInfo
    (UInt16 btLibRefNum, UInt16 index,
    BtLibDeviceAddressTypePtr addrP,
    Char *nameBuffer,
    UInt8 *nameBufferSize,
    BtLibClassOfDeviceType *cod,
    Boolean *persistent)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *index*
Index of the record.

← *addrP*
Bluetooth address of remote device.

← *nameBuffer*
Pointer to buffer to store user-friendly name of remote device. You must allocate this buffer. Provide a `NULL` pointer if the user-friendly name is not needed.

↔ *nameBufferSize*
Size of the `nameBuffer` buffer on entry. On exit, the size of the name.

← *cod*
Pointer to a `BtLibClassOfDeviceType` representing the class of the device. You must allocate this structure. Provide a `NULL` pointer if the device class is not needed.

← *lastConnected*
The date since the device last connected. This date is measured in seconds since midnight January 1, 1904. Provide a `NULL` pointer if the date of last connection is not needed.

← *persistent*
If `true`, the device is bonded and can connect to the local device without authentication. If `false`, the device is paired but not bonded—it will need to reauthenticate if it connects

again. Provide a `NULL` pointer if this information is not needed.

Returns Returns `btLibErrNoError` if successful. Returns `dmErrIndexOutOfRange` if a record with the specified index could not be found.

See Also `BtLibSecurityFindTrustedDeviceRecord()`

BtLibSecurityNumTrustedDeviceRecords Function

Purpose Return the number of bonded devices in the device database or return the total number of devices in the device database.

Declared In `BtLib.h`

Prototype

```
UInt16 BtLibSecurityNumTrustedDeviceRecords
    (UInt16 btLibRefNum, Boolean persistentOnly)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *persistentOnly*
`true` to obtain the total number of bonded devices in the database. These are the same devices that appear in the trusted devices list. `false` to obtain the total number of devices in the device database. This includes the devices that are bonded and the devices that are paired but not bonded.

Returns Returns the requested number of device records.

See Also `BtLibSecurityFindTrustedDeviceRecord()`, `BtLibSecurityGetTrustedDeviceRecordInfo()`

BtLibSecurityRemoveTrustedDevice Record Function

Purpose Remove a device record from the device database.

Declared In `BtLib.h`

Prototype

```
Err BtLibSecurityRemoveTrustedDeviceRecord
    (UInt16 btLibRefNum, UInt16 index)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *index*
Index of the record to remove.

Returns Returns `btLibErrNoError` if successful. Returns `dmErrIndexOutOfRange` if a record with the specified index could not be found.

See Also `BtLibSecurityFindTrustedDeviceRecord()`

Utility Functions

This section describes functions and macros that are useful when working with the Bluetooth library.

BtLibAddrAToBtd Function

Purpose Convert an ASCII string a Bluetooth device address in colon-separated form to a 48-bit `BtLibDeviceAddressType`.

Declared In `BtLib.h`

Prototype

```
Err BtLibAddrAToBtd (UInt16 btLibRefNum,
    const Char *addressString,
    BtLibDeviceAddressType *btDevP)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *addressString*
String containing ASCII colon-separated Bluetooth device address.

← *btDevP*
Pointer to a BtLibDeviceAddressType to store the converted device address.

Returns Returns btLibErrNoError to indicate that the conversion was successful.

See Also BtLibAddrBtdToA()

BtLibAddrBtdToA Function

Purpose Convert 48-bit BtLibDeviceAddressType to an ASCII string in colon-separated form.

Declared In BtLib.h

Prototype

```
Err BtLibAddrBtdToA (UInt16 btLibRefNum,
    BtLibDeviceAddressType *btDevP, Char *spaceP,
    UInt16 spaceSize)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *btDevP*
Address of a Bluetooth device. This parameter must not be NULL.

← *spaceP*
Pointer to a buffer to store the ASCII formatted Bluetooth devices address upon return. This parameter must not be NULL.

→ *spaceSize*
Size of the spaceP buffer, in bytes. Must be at least 18.

Returns Returns btLibErrNoError if successful. Returns btLibErrParamErr if

- btDevP is NULL
- spaceP is NULL
- spaceSize is less than 18, the number of bytes required to store the ASCII formatted address

See Also BtLibAddrAToBtd()

BtLibL2CapHToNL Macro

Purpose Converts a 32-bit value from host to L2CAP byte order. L2CAP byte order is little endian.

Declared In `BtLib.h`

Prototype
```
#define BtLibL2CapHToNL (value)
```

Parameters → *value*
32-bit value to convert.

Returns Returns `value` in L2CAP byte order.

See Also `BtLibL2CapHToNS()`, `BtLibL2CapNToHL()`, `BtLibL2CapNToHS()`

BtLibL2CapHToNS Macro

Purpose Converts a 16-bit value from host to L2CAP byte order. L2CAP byte order is little endian.

Declared In `BtLib.h`

Prototype
```
#define BtLibL2CapHToNS (value)
```

Parameters → *value*
16-bit value to convert.

Returns Returns `value` in L2CAP byte order.

See Also `BtLibL2CapHToNL()`, `BtLibL2CapNToHS()`, `BtLibSdpNToHL()`

BtLibL2CapNToHL Macro

Purpose Converts a 32-bit value from L2CAP to host byte order. L2CAP byte order is little endian.

Declared In `BtLib.h`

Prototype
```
#define BtLibL2CapNToHL (value)
```

Parameters → *value*
32-bit value to convert.

Returns Returns `value` in host byte order.

See Also `BtLibL2CapNToHS()`, `BtLibL2CapHToNL()`, `BtLibL2CapHToNS()`

BtLibL2CapNToHS Macro

Purpose Converts a 16-bit value from L2CAP to host byte order. L2CAP byte order is little endian.

Declared In `BtLib.h`

Prototype

```
#define BtLibL2CapNToHS (value)
```

Parameters → *value*
16-bit value to convert.

Returns Returns `value` in host byte order.

See Also `BtLibL2CapNToHL()`, `BtLibL2CapHToNS()`, `BtLibL2CapHToNL()`

BtLibRfCommHToNL Macro

Purpose Converts a 32-bit value from host to RFCOMM byte order. RFCOMM byte order is big endian.

Declared In `BtLib.h`

Prototype

```
#define BtLibRfCommHToNL (value)
```

Parameters → *value*
32-bit value to convertx

Returns Returns `value` in RFCOMM byte order.

See Also BtLibRfCommHToNS(), BtLibRfCommNToHL(), BtLibRfCommNToHS()

BtLibRfCommHToNS Macro

Purpose Converts a 16-bit value from host to RFCOMM byte order. RFCOMM byte order is big endian.

Declared In `BtLib.h`

Prototype

```
#define BtLibRfCommHToNS (value)
```

Parameters → *value*
16-bit value to convertx

Returns Returns `value` in RFCOMM byte order.

See Also BtLibRfCommHToNL(), BtLibRfCommHToNS(), BtLibRfCommNToHL()

BtLibRfCommNToHL Macro

Purpose Converts a 32-bit value from RFCOMM to host byte order. RFCOMM byte order is big endian.

Declared In `BtLib.h`

Prototype

```
#define BtLibRfCommNToHL (value)
```

Parameters → *value*
32-bit value to convertx

Returns Returns `value` in host byte order.

See Also BtLibRfCommNToHS(), BtLibRfCommHToNL(), BtLibRfCommHToNS()

BtLibRfCommNToHS Macro

Purpose Converts a 16-bit value from RFCOMM to host byte order. RFCOMM byte order is big endian.

Declared In `BtLib.h`

Prototype

```
#define BtLibRfCommNToHS (value)
```

Parameters → *value*
16-bit value to convertx

Returns Returns `value` in host byte order.

See Also `BtLibRfCommNToHL()`, `BtLibRfCommHToNS()`, `BtLibRfCommHToNL()`

BtLibSdpHToNL Macro

Purpose Converts a 32-bit value from host to Service Discovery Protocol (SDP) byte order. SDP byte order is big endian.

Declared In `BtLib.h`

Prototype

```
#define BtLibSdpHToNL (value)
```

Parameters → *value*
32-bit value to convertx

Returns Returns `value` in SDP byte order.

See Also `BtLibSdpHToNS()`, `BtLibSdpNToHL()`, `BtLibSdpNToHS()`

BtLibSdpHToNS Macro

Purpose Converts a 16-bit value from host to Service Discovery Protocol (SDP) byte order. SDP byte order is big endian.

Declared In `BtLib.h`

Prototype

```
#define BtLibSdpHToNS (value)
```

Parameters → *value*
16-bit value to convertx

Returns Returns `value` in SDP byte order.

See Also `BtLibSdpHToNL()`, `BtLibSdpNToHS()`, `BtLibSdpNToHL()`

BtLibSdpNToHL Macro

Purpose Converts a 32-bit value from Service Discovery Protocol (SDP) to host byte order. SDP byte order is big endian.

Declared In `BtLib.h`

Prototype

```
#define BtLibSdpNToHL (value)
```

Parameters → *value*
32-bit value to convertx

Returns Returns `value` in host byte order.

See Also `BtLibSdpNToHS()`, `BtLibSdpHToNL()`, `BtLibSdpHToNS()`

BtLibSdpNToHS Macro

Purpose Converts a 16-bit value from Service Discovery Protocol (SDP) to host byte order. SDP byte order is big endian.

Declared In `BtLib.h`

Prototype

```
#define BtLibSdpNToHS (value)
```

Parameters → *value*
16-bit value to convertx

Returns Returns `value` in host byte order.

See Also `BtLibSdpNToHL()`, `BtLibSdpHToNS()`, `BtLibSdpHToNL()`

82

Bluetooth Library: Management

The management API of the Bluetooth library supports the lower levels of the Bluetooth specification, specifically the radio, baseband, and Link Manager Protocol specifications. This chapter presents reference material for the management API:

`Bluetooth Management Data Structures`
: This section lists some of the more important types used by the Bluetooth library management functions.

`Management Callback Events`
: This section lists the management callback events. Most of the management functions are asynchronous. In other words, they start operations and return before the operations complete. To signal the application that management operations have completed, the Bluetooth library generates **management callback events** by calling a callback function.

`Management Event Status Codes`
: When a management event is generated, the `status` field of the associated `BtLibManagementEventType` provides information about why the event occurred. This section explains what these codes mean.

`Library Management Functions`
: This section describes the functions that open and close the shared library.

`Management Functions`
: The management functions handle the lower levels of the Bluetooth specification, specifically the radio, baseband, and Link Manager Protocol specifications. These functions perform tasks that include discovering devices, working with Asynchronous Connectionless (ACL) links and piconets, and maintaining global settings for the Bluetooth library.

Application-Defined Functions
: This section describes the callback functions that handle management events.

The header file `BtLib.h` declares the Bluetooth library functions and macros. The header file `BtLibTypes.h` declares the data structures that you use with those functions and macros. For more information about using the Bluetooth library, see the *Palm OS Programmer's Companion Supplement: Bluetooth*.

Bluetooth Management Data Structures

This section lists some of the more important types used by the Bluetooth library management functions.

BtLibAccessibleModeEnum Enum

Purpose The `BtLibAccessibleModeEnum` enum specifies a device's accessibility modes. See the "Generic Access Profile" chapter of the *Specification of the Bluetooth System* for more information about accessibility.

Prototype

```
typedef enum {
  btLibNotAccessible = 0x00,
  btLibConnectableOnly = 0x02,
  btLibDiscoverableAndConnectable = 0x03
} BtLibAccessibleModeEnum;
```

Constants

`btLibConnectableOnly`
: The device responds to a page but not an inquiry.

`btLibDiscoverableAndConnectable`
: The device responds to both a page and an inquiry.

`btLibNotAccessible`
: The device does not respond to a page or an inquiry.

BtLibClassOfDeviceType Typedef

Purpose The `BtLibClassOfDeviceType` type represents the class of the device and the services it supports.

Prototype `typedef UInt32 BtLibClassOfDeviceType;`

A device can support multiple services but only belongs to a single class. The class is specified in two parts: the major class, which broadly classifies the type of device, and the minor class, which together with the major class specifies the type of device in more detail.

An example is a simple cellular telephone. It provides Telephony services. Its major device class is Phone, and its minor device class is Cellular.

The *Bluetooth Assigned Numbers* specification defines a device class as having three bit fields. One field specifies the major service classes supported by the device. Another field specifies the major device class. The third field specifies the minor device class.

The constants provided here allow you to construct a device class that conforms to the Bluetooth specification. You simply perform a logical OR of the constants representing the service classes the device supports, the constant representing the device's major class, and the constant representing the device's minor class.

For example, device class of the simple cellular telephone can be computed as follows:

```
cellPhoneCOD = btLibCOD_Telephony |
  btLibCOD_Major_Phone |
  BtLibCOD_Minor_Phone_Cellular;
```

Constants are also provided to mask the individual bit fields in a device class.

Major Service Classes

These constants define the Bluetooth major service classes. The service classes are described in the *Specification of the Bluetooth System*.

Constants `btLibCOD_LimitedDiscoverableMode`

`btLibCOD_Networking`

`btLibCOD_Rendering`

`btLibCOD_Capturing`

`btLibCOD_ObjectTransfer`

`btLibCOD_Audio`

`btLibCOD_Telephony`

`btLibCOD_Information`

Major Device Classes

These constants define the Bluetooth major device classes. The major device classes are described in the *Specification of the Bluetooth System*.

Constants

`btLibCOD_Major_Misc`

`btLibCOD_Major_Computer`

`btLibCOD_Major_Phone`

`btLibCOD_Major_Lan_Access_Point`

`btLibCOD_Major_Audio`

`btLibCOD_Major_Peripheral`

`btLibCOD_Major_Imaging`

`btLibCOD_Major_Unclassified`

Computer Minor Device Classes

These constants define the minor device classes associated with the computer major class. They are described in the *Bluetooth Assigned Numbers* specification.

Constants

`btLibCOD_Minor_Comp_Unclassified`

`btLibCOD_Minor_Comp_Desktop`

`btLibCOD_Minor_Comp_Server`

`btLibCOD_Minor_Comp_Laptop`

`btLibCOD_Minor_Comp_Handheld`

`btLibCOD_Minor_Comp_Palm`

Phone Minor Device Classes

These constants define the minor device classes are associated with the computer major class. They are described in the *Bluetooth Assigned Numbers* specification.

Constants

`btLibCOD_Minor_Phone_Unclassified`

`btLibCOD_Minor_Phone_Cellular`

`btLibCOD_Minor_Phone_ISDN`

`btLibCOD_Minor_Phone_Cordless`

`btLibCOD_Minor_Phone_Smart`

`btLibCOD_Minor_Phone_Modem`

LAN Access Point Minor Device Classes

These constants define load factors for the LAN access point major device class. LAN access point load factors are described in more detail in the *Bluetooth Assigned Numbers* specification.

Constants

`btLibCOD_Minor_Lan_0`
Fully available

`btLibCOD_Minor_Lan_17`
1-17% utilized

`btLibCOD_Minor_Lan_33`
17-33% utilized

`btLibCOD_Minor_Lan_50`
33-50% utilized

`btLibCOD_Minor_Lan_67`
50-67% utilized

`btLibCOD_Minor_Lan_83`
67-83% utilized

`btLibCOD_Minor_Lan_99`
83-99% utilized

`btLibCOD_Minor_Lan_NoService`
Fully utilized

Audio Minor Device Classes

These constants define the minor classes associated with the audio major class. They are described in more detail in the *Bluetooth Assigned Numbers* specification.

Constants

`btLibCOD_Minor_Audio_Unclassified`

`btLibCOD_Minor_Audio_Headset`

`btLibCOD_Minor_Audio_HandFree`

`btLibCOD_Minor_Audio_MicroPhone`

`btLibCOD_Minor_Audio_LoudSpeaker`

`btLibCOD_Minor_Audio_HeadPhone`

`btLibCOD_Minor_Audio_PortableAudio`

`btLibCOD_Minor_Audio_CarAudio`

`btLibCOD_Minor_Audio_SetTopBox`

`btLibCOD_Minor_Audio_HifiAudio`

`btLibCOD_Minor_Audio_VCR`

`btLibCOD_Minor_Audio_VideoCamera`

`btLibCOD_Minor_Audio_CamCorder`

`btLibCOD_Minor_Audio_VideoMonitor`

`btLibCOD_Minor_Audio_VideoDisplayAndLoudSpeaker`

`btLibCOD_Minor_Audio_VideoConf`

`btLibCOD_Minor_Audio_GameToy`

Peripheral Minor Device Classes

These constants define the minor classes associated with the peripheral major class. They are described in more detail in the *Bluetooth Assigned Numbers* specification.

Constants

`btLibCOD_Minor_Peripheral_Unclassified`

`btLibCOD_Minor_Peripheral_Keyboard`

`btLibCOD_Minor_Peripheral_Pointing`

`btLibCOD_Minor_Peripheral_Combo`

Peripheral Minor Device Subclasses

These constants define further subclasses associated with the peripheral minor classes. They are described in more detail in the *Bluetooth Assigned Numbers* specification.

Constants

`btLibCOD_Minor_Peripheral_Joystick`

`btLibCOD_Minor_Peripheral_GamePad`

`btLibCOD_Minor_Peripheral_RemoteControl`

`btLibCOD_Minor_Peripheral_Sensing`

`btLibCOD_Minor_Peripheral_DigitizerTablet`

`btLibCOD_Minor_Peripheral_CardReader`

Imaging Minor Device Classes

These constants define the minor classes associated with the imaging major class. They are described in more detail in the *Bluetooth Assigned Numbers* specification.

Constants

`btLibCOD_Minor_Imaging_Unclassified`

`btLibCOD_Minor_Imaging_Display`

`btLibCOD_Minor_Imaging_Camera`

`btLibCOD_Minor_Imaging_Scanner`

`btLibCOD_Minor_Imaging_Printer`

Masks

These constants define bit masks to isolate certain fields of the device class.

Constants

`btLibCOD_Service_Mask`
: A mask to isolate the major service class field from the other fields of the device class.

`btLibCOD_Major_Mask`
: A mask to isolate the major device class field from the other fields of the device class.

`btLibCOD_Minor_Mask`
: A mask to isolate the minor device class field from the other fields of the device class.

`btLibCOD_ServiceAny`
: Used as a device filter for the `BtLibDiscoverMultipleDevices()` and `BtLibDiscoverSingleDevice()` functions. With this filter, devices providing any service appear in the device list. Same as `btLibCOD_Service_Mask`.

`btLibCOD_Major_Any`
: Used as a device filter for the `BtLibDiscoverMultipleDevices()` and `BtLibDiscoverSingleDevice()` functions. With this filter, devices in any major device class appear in the device list. Same as `btLibCOD_Major_Mask`.

`btLibCOD_Minor_Any`
: Used as a device filter for the `BtLibDiscoverMultipleDevices()` and `BtLibDiscoverSingleDevice()` functions. With this filter, devices in any minor device class appear in the device list. Same as `btLibCOD_Minor_Mask`.

`btLibCOD_Minor_Comp_Any`
: Used as a device filter for the `BtLibDiscoverMultipleDevices()` and `BtLibDiscoverSingleDevice()` functions. When this filter is used in conjunction with `btLibCOD_Major_Computer`, all devices broadcasting themselves as computers appear in the device list. Same as `btLibCOD_Minor_Any`.

`btLibCOD_Minor_Phone_Any`
: Used as a device filter for the `BtLibDiscoverMultipleDevices()` and `BtLibDiscoverSingleDevice()` functions. When this filter is used in conjunction with `btLibCOD_Major_Phone`, all devices broadcasting themselves as phones appear in the device list. Same as `btLibCOD_Minor_Any`.

`btLibCOD_Minor_LAN_Any`
: Used as a device filter for the `BtLibDiscoverMultipleDevices()` and `BtLibDiscoverSingleDevice()` functions. When this filter is used in conjunction with `btLibCOD_Major_Lan_Access_Point`, all devices

broadcasting themselves as LAN access points appear in the device list. Same as `btLibCOD_Minor_Any`.

`btLibCOD_Minor_Audio_Any`
: Used as a device filter for the `BtLibDiscoverMultipleDevices()` and `BtLibDiscoverSingleDevice()` functions. When this filter is used in conjunction with `btLibCOD_Major_Audio`, all devices broadcasting themselves as audio devices appear in the device list. Same as `btLibCOD_Minor_Any`.

BtLibConnectionRoleEnum Enum

Purpose The `BtLibConnectionRoleEnum` enum specifies all the connection roles a device can have. A device can either be a master or a slave.

Prototype

```
typedef enum {
  btLibMasterRole,
  btLibSlaveRole
} BtLibConnectionRoleEnum;
```

Constants

`btLibMasterRole`
: The device is a master.

`btLibSlaveRole`
: The device is a slave.

BtLibDeviceAddressType Struct

Purpose The `BtLibDeviceAddressType` structure represents a 48-bit Bluetooth device address.

Prototype

```
#define btLibDeviceAddressSize 6

typedef struct BtLibDeviceAddressType {
    UInt8 address[btLibDeviceAddressSize];
} BtLibDeviceAddressType;
```

BtLibFriendlyNameType Struct

Purpose The `BtLibFriendlyNameType` structure contains the user-friendly name of a device.

Prototype

```
typedef struct BtLibFriendlyNameType {
  UInt8  *name;
  UInt8  nameLength;
} BtLibFriendlyNameType,
  *BtLibFriendlyNameTypePtr;
```

Fields

`name`
: Array of characters, encoded according to the UTF-8 standard, containing the user-friendly name of the device. This array is not null-terminated.

`nameLength`
: The number of characters in the user-friendly name. The maximum size is 249 characters.

BtLibManagementEventType Struct

Purpose The `BtLibManagementEventType` structure contains detailed information regarding a management callback event. All management events have some common data. Most management events have data specific to those events. The specific data uses a union that is part of the `BtLibManagementEvent` data structure.

Prototype

```
typedef struct _BtLibManagementEventType {
  BtLibManagementEventEnum event;
  Err status;
  union {
    ...
  } eventData;
} BtLibManagementEventType;
```

Fields

`event`
: A `BtLibManagementEventEnum` enum member that indicates which management event has occurred. See Management Callback Events.

`status`
: Status of the event. The Management Callback Events section gives more details about how to interpret this field for specific events.

`eventData`
: Data associated with the event. The member of this union that is valid depends on the event. See Management Callback Events for more information.

A `BtLibManagementEventType` object is passed as the first argument of the `BtLibManagementCallback()` callback function.

Management Callback Events

The management functions of the Bluetooth library support the lower levels of the Bluetooth specification, specifically the radio, baseband, and Link Manager Protocol specifications. Most of the management functions are asynchronous. In other words, they start operations and return before the operations complete. To signal the application that management operations have completed, the Bluetooth library generates **management callback events** by calling a callback function.

You specify the callback function using the `BtLibRegisterManagementNotification()` function. When an event occurs, the callback function is called with two parameters: a pointer to a `BtLibManagementEventType` structure and a pointer to a user-defined structure.

The `BtLibManagementEventType` structure contains an `event` field, which indicates the reason the callback is called, a `status` field, which contains status information associated with the event, and a union of several structures. The member of the union that is valid depends on the event. The meaning of the events is described in the following sections.

For more information about the `status` field, see Management Event Status Codes.

btLibManagementEventAccessibilityChange

The accessibility mode of the local device has changed.

For this event, the `eventData` field contains the following field:

```
BtLibAccessibleModeEnum accessible;
```

This `BtLibAccessibleModeEnum` represents the new accessibility mode of the local device.

This event can result from calling `BtLibOpen()`, `BtLibPiconetCreate()`, or `BtLibSetGeneralPreference()`.

btLibManagementEventAclConnectInbound

A remote device has established an ACL link to the local device.

For this event, the `eventData` field contains the following field:

```
BtLibDeviceAddressType bdAddr;
```

This `BtLibDeviceAddressType` contains the address of the remote device.

btLibManagementEventAclConnectOutbound

An ACL link has been established with a remote device.

If the `status` field contains `btLibErrNoError`, the ACL link is connected, and the remote device address can be found in the `eventData` field. Otherwise the connection failed and the `status` field indicates the reason for the failure. See Management Event Status Codes for more information.

For this event, the `eventData` field contains the following field:

```
BtLibDeviceAddressType bdAddr;
```

This `BtLibDeviceAddressType` contains the address of the remote device.

This event can result from calling `BtLibLinkConnect()`.

btLibManagementEventAclDisconnect

An ACL link has been disconnected. The `status` field indicates the reason the link was disconnected. See Management Event Status Codes.

For this event, the `eventData` field contains the following field:

```
BtLibDeviceAddressType bdAddr;
```

This `BtLibDeviceAddressType` contain the address of the disconnected device.

This event can result from calling `BtLibLinkDisconnect()` or `BtLibPiconetDestroy()`.

btLibManagementEventAuthentication Complete

The authentication of a remote device has completed.

For this event, the `eventData` field contains the following field:

```
BtLibDeviceAddressType bdAddr;
```

This `BtLibDeviceAddressType` contains the address of the remote device.

If the authentication is successful, the `status` field contains `btLibErrNoError`. If the user cancels the passkey request, the `status` field contains `btLibErrCanceled`. Otherwise, the `status` field indicates the reason the authentication failed. See Management Event Status Codes.

This event can result from calling `BtLibLinkSetState()`.

btLibManagementEventEncryptionChange

Purpose Encryption for a link has been enabled or disabled.

For this event, the `eventData` field contains the following structure:

Prototype

```
struct {
  BtLibDeviceAddressType bdAddr;
  Boolean enabled;
} encryptionChange;
```

Fields

`bdAddr`
: A `BtLibDeviceAddressType` containing the address of the remote device.

`enabled`
: `true` when encryption for the link has been enabled; `false` otherwise.

This event can result from calling `BtLibLinkSetState()`.

btLibManagementEventInquiryCanceled

The device inquiry has been canceled because the application called `BtLibCancelInquiry()`.

btLibManagementEventInquiryComplete

The device inquiry started with the `BtLibStartInquiry()` function has completed.

btLibManagementEventInquiryResult

Purpose

A remote device has responded to an inquiry that was started with the `BtLibStartInquiry()` function.

For this event, the `eventData` field contains the following structure:

Prototype

```
struct {
  BtLibDeviceAddressType bdAddr;
  BtLibClassOfDeviceType classOfDevice;
} inquiryResult;
```

Fields

`bdAddr`
: A `BtLibDeviceAddressType` containing the address of the remote device.

`classOfDevice`
: A `BtLibClassOfDeviceType` representing the class of the remote device.

btLibManagementEventLocalNameChange

Purpose The user-friendly name of the local device has changed.

For this event, the `eventData` field contains the following structure:

Prototype

```
struct {
  BtLibDeviceAddressType bdAddr;
  BtLibFriendlyNameType name;
} nameResult;
```

Fields

`bdAddr`
: A `BtLibDeviceAddressType` containing the address of the local device.

`name`
: A `BtLibFriendlyNameType` containing the new name.

This event can result from calling `BtLibOpen()`.

btLibManagementEventModeChange

Purpose A slave has changed its mode. A slave can be in active, sniff, hold, or park mode.

For this event, the `eventData` field contains the following structure:

Prototype

```
struct {
  BtLibDeviceAddressType bdAddr;
  BtLibLinkModeEnum curMode;
  UInt16 interval;
} modeChange;
```

Fields

`bdAddr`
: A `BtLibDeviceAddressType` containing the address of the remote device.

`curMode`
: A `BtLibLinkModeEnum` indicating the new mode of remote device

`interval`
: The time in units of 0.625 ms the remote device will stay in the new mode, if applicable. The time period is a standard time period in the Bluetooth specification.

BtLibLinkModeEnum

Purpose

The `BtLibLinkModeEnum` enum specifies the modes a slave can have. According to the *Specification of the Bluetooth System*, a slave can be in active, sniff, hold, or park mode. However, the Bluetooth library only supports the hold and active modes.

Prototype

```
typedef enum {
  btLibSniffMode,
  btLibHoldMode,
  btLibParkMode,
  btLibActiveMode
} BtLibLinkModeEnum;
```

Constants

`btLibActiveMode`
: The slave is active.

`btLibHoldMode`
: The slave is in hold mode.

`btLibParkMode`
: The slave is in park mode. This mode is not currently supported.

`btLibSniffMode`
: The slave is in sniff mode. This mode is not currently supported.

btLibManagementEventNameResult

Purpose

A remote device name request has completed. If the `status` field is `btLibErrNoError`, the name is available. Otherwise, the name request failed, and the `status` field indicates the reason for the failure. See Management Event Status Codes.

For this event, the `eventData` field contains the following structure:

Prototype

```
struct {
  BtLibDeviceAddressType bdAddr;
  BtLibFriendlyNameType name;
} nameResult;
```

Fields

`bdAddr`
: A `BtLibDeviceAddressType` containing the address of the remote device.

`name`
: A `BtLibFriendlyNameType` containing the name of the remote device.

The `BtLibGetRemoteDeviceName()` function is used to start a remote device name request.

btLibManagementEventPairingComplete

Pairing has successfully completed and the link is authenticated. The status code for this event is set to `btLibErrNoError`, indicating that the link key was stored successfully.

If pairing fails, no event is sent (there is no event sent by the radio module indicating failure).

btLibManagementEventPasskeyRequest

A remote device has requested a passkey. Your application does not have to respond to this request—the Bluetooth library automatically handles it.

For this event, the `eventData` field contains the following field:

```
BtLibDeviceAddressType bdAddr;
```

This `BtLibDeviceAddressType` contains the address of the remote device.

Because a passkey can be requested during or after a link is established, consider disabling any failure timers while the passkey

dialog is up. The btLibManagementEventPasskeyRequestComplete event signals the completion of the passkey entry.

btLibManagementEventPasskeyRequestComplete

A passkey request has been processed. The status code for this event is set to `btLibErrNoError` if the passkey was entered or `btLibErrCanceled` if passkey entry was cancelled. Note that this event does *not* tell you that the authentication completed.

btLibManagementEventPiconetCreated

The piconet has been created. This event can result from calling `BtLibPiconetCreate()`.

btLibManagementEventPiconetDestroyed

The piconet has been destroyed. This event can result from calling `BtLibPiconetDestroy()`.

btLibManagementEventRadioState

This event is generated when the Bluetooth radio changes state. The radio changes state when the radio is disconnected, the power is turned on or off, the radio resets, or the radio fails to initialize. The status code for this event explains why the event gets generated.

Status Codes

`btLibErrRadioInitialized`
: The Bluetooth radio has initialized successfully. You can now call management functions.

`btLibErrRadioInitFailed`
: The Bluetooth radio failed to initialize. The application can assume all pending Bluetooth operations have failed.

However, some pending operations will still generate events and modify memory supplied by the application.

To try to initialize the radio again, you need to close the library and reopen it.

`btLibErrRadioFatal`
: A fatal radio error occurred. This usually signifies that the host has lost contact with the radio, for example, when the user disconnects the radio, or the device turns off. The application can assume that all pending operations have failed. However, some pending operations will still generate events and modify memory supplied by the application.

When a fatal radio error occurs, the Bluetooth stack resets the radio and tries once to reinitialize it, which generates another `btLibManagementEventRadioState` event with a status code of `btLibErrRadioInitialized`, or `btLibErrRadioInitFailed` depending on whether or not the initialization succeeded.

`btLibErrRadioSleepWake`
: The radio was reset because the device went to sleep. The application can assume all pending operations have failed. However, some pending operations will still generate events and modify memory supplied by the application.

The Bluetooth stack resets the radio and tries once to reinitialize it, which generates another `btLibManagementEventRadioState` event with a status code of `btLibErrRadioInitialized`, or `btLibErrRadioInitFailed` depending on whether or not the initialization succeeded.

This event can result from calling `BtLibOpen()`.

btLibManagementEventRoleChange

Purpose The master and slave devices for a link have switched roles.

For this event, the `eventData` field contains the following structure:

Prototype

```
struct {
  BtLibDeviceAddressType bdAddr;
  BtLibConnectionRoleEnum newRole;
} roleChange;
```

Fields

`bdAddr`
: A `BtLibDeviceAddressType` containing the address of the remote device.

`newRole`
: A `BtLibConnectionRoleEnum` representing the new role of the local device.

Management Event Status Codes

When a management event is generated, the `status` field of the associated `BtLibManagementEventType` provides information about why the event occurred. The following status codes can occur with a management event.

`btLibErrNoError`
: Success.

`btLibMeStatusAuthenticateFailure`
: Authentication failure

`btLibMeStatusCommandDisallowed`
: Command disallowed

`btLibMeStatusConnnectionTimeout`
: Connection timeout

`btLibMeStatusHardwareFailure`
: Hardware Failure

`btLibMeStatusHostTimeout`
: Host timeout

`btLibMeStatusInvalidHciParam`
: Invalid HCI command parameters

`btLibMeStatusInvalidLmpParam`
: Invalid LMP Parameters

`btLibMeStatusLimitedResources`
: Host rejected due to limited resources

`btLibMeStatusLmpResponseTimeout`

`btLibMeStatusLmpTransdCollision`

`btLibMeStatusLmpPduNotAllowed`

`btLibMeStatusLocalTerminated`
Terminated by local host

`btLibMeStatusLowResources`
Other end terminated due to low resources

`btLibMeStatusMaxAclConnections`
Max number of ACL connections to a device

`btLibMeStatusMaxConnections`
Max number of connections

`btLibMeStatusMaxScoConnections`
Max number of SCO connections to a device

`btLibMeStatusMemoryFull`
Memory full

`btLibMeStatusMissingKey`
Missing key

`btLibMeStatusNoConnection`
No connection

`btLibMeStatusPageTimeout`
Page timeout

`btLibMeStatusPairingNotAllowed`
Pairing not allowed

`btLibMeStatusPersonalDevice`
Host rejected (remote is personal device)

`btLibMeStatusPowerOff`
Other end terminated (about to power off)

`btLibMeStatusRepeatedAttempts`
Repeated attempts

`btLibMeStatusRoleChangeNotAllowed`
Change not allowed

`btLibMeStatusScoAirModeRejected`
SCO Air Mode Rejected

`btLibMeStatusScoIntervalRejected`
SCO Interval Rejected

`btLibMeStatusScoOffsetRejected`
SCO Offset Rejected

`btLibMeStatusSecurityError`
Host rejected due to security reasons

`btLibMeStatusUnknownHciCommand`
Unknown HCI Command

`btLibMeStatusUnknownLmpPDU`
Unknown LMP PDU

`btLibMeStatusUnspecifiedError`
Unspecified Error

`btLibMeStatusUnsupportedFeature`
Unsupported feature or parameter value

`btLibMeStatusUnsupportedLmpParam`
Unsupported LMP Parameter Value

`btLibMeStatusUnsupportedRemote`
Unsupported Remote Feature

`btLibMeStatusUserTerminated`
Other end terminated (user)

Library Management Functions

This section describes the general Bluetooth library management functions.

BtLibClose Function

Purpose Close the Bluetooth library.

Declared In `BtLib.h`

Prototype

```
Err BtLibClose (UInt16 btLibRefNum)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

Returns Returns `btLibErrNoError` if the library successfully closes. Returns `btLibErrNotOpen` if the referenced Bluetooth library was not open.

Comments Applications must call this function when they no longer need the Bluetooth library. If the Bluetooth library open count is one, this function closes existing connections, saves the current accessibility mode, sets the accessible mode according to the preferences panel, and shuts down the library. If the Bluetooth library open count is greater than one, this function decrements the open count.

See Also `BtLibOpen()`

BtLibOpen Function

Purpose Open and initialize the Bluetooth library.

Declared In `BtLib.h`

Prototype

```
Err BtLibOpen (UInt16 btLibRefNum,
    Boolean allowStackToFail)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *allowStackToFail*
If true, opens the library even if the stack or radio fails to initialize. Otherwise, does not open the library if the stack or radio fails to initialize.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrAlreadyOpen`
: This status code is not really an error. It is returned if the library was already open. In this case, the open count is incremented.

`btLibErrInUseByService`
: The library is currently in use by a Bluetooth service.

`btLibErrOutOfMemory`
: Not enough memory available to open the library.

`btLibErrRadioInitFailed`
: The Bluetooth stack or radio could not be initialized. If `allowStackToFail` is true, the library still opens after this error occurs.

Applications must call this function before using the Bluetooth library. If the Bluetooth library is not already open, `BtLibOpen()` opens the library, initializes it, and starts up the protocol stack component of the library. Otherwise it increments its open count.

The `allowStackToFail` parameter allows the library to be opened even if the Bluetooth stack or radio fails to initialize. It is useful for applications that only want to use the Bluetooth library's utility functions but not the radio. However, any application that needs to communicate with the radio must set `allowStackToFail` to `false`.

Once you've opened the Bluetooth library, you can then register your Bluetooth event handler by calling `BtLibRegisterManagementNotification()`. Once you've done that, your application will begin receiving events.

This function generates three events: a `btLibManagementEventRadioState` event with a status of `btLibErrRadioInitialized`, a `btLibManagementEventLocalNameChange` event indicating the local name of the device, and a `btLibManagementEventAccessibilityChange` event indicating the accessibility of the device.

Your application will not miss these events between calling `BtLibOpen()` and your call to the `BtLibRegisterManagementNotification()` function because

the Bluetooth stack is driven by the application's event loop; events are only sent to your application during the event loop.

See Also `BtLibClose()`

Management Functions

The management functions handle the lower levels of the Bluetooth specification, specifically the radio, baseband, and Link Manager Protocol specifications. These functions perform tasks that include discovering devices, working with Asynchronous Connectionless (ACL) links and piconets, and maintaining global settings for the Bluetooth library.

BtLibCancelInquiry Function

Purpose Cancel a Bluetooth inquiry in process.

Declared In `BtLib.h`

Prototype

```
Err BtLibCancelInquiry (UInt16 btLibRefNum)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

Returns Returns one of the following values:

`btLibErrNoError`
The inquiry process was canceled before it started.

`btLibErrPending`
The cancellation is pending. When it succeeds, notification will be provided through a management callback event.

`btLibErrInProgress`
The inquiry is already being canceled.

`btLibErrNotInProgress`
No inquiry is in progress to be canceled.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments The function cancels inquiries initiated by `BtLibStartInquiry()`. The `btLibManagementEventInquiryCanceled` callback event indicates that the cancellation has completed.

A Bluetooth discovery initiated using either `BtLibDiscoverSingleDevice()` or `BtLibDiscoverMultipleDevices()` cannot be canceled with this function. Only the user can cancel these inquiries by tapping the Cancel button.

See Also BtLibStartInquiry()

BtLibDiscoverMultipleDevices Function

Purpose Discover all available devices, present them in the user interface, and allow the user to select one or more of these devices.

Declared In `BtLib.h`

Prototype

```
Err BtLibDiscoverMultipleDevices
    (UInt16 btLibRefNum, Char *instructionTxt,
    Char *buttonTxt,
    BtLibClassOfDeviceType *deviceFilterList,
    UInt8 deviceFilterListLen,
    UInt8 *numDevicesSelected,
    Boolean addressAsName, Boolean showLastList)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *instructionTxt*
Text displayed at the top of the selection box. Pass `NULL` to display the default text. The default text is "Select one or more devices:"

→ *buttonTxt*
Text for the OK button. Pass `NULL` to display the default text. The default button text is "OK"

→ *deviceFilterList*
Array of `BtLibClassOfDeviceType`s. This function presents to the user only the remote devices whose class matches a class in this list. If `deviceFilterList` is `NULL`, this function presents to the user all discovered devices.

→ *deviceFilterListLen*
Number of elements in `deviceFilterList`.

← *numDevicesSelected*
Number of selected devices. To obtain the actual device list, use the `BtLibGetSelectedDevices` function.

→ *addressAsName*
If `true`, display the Bluetooth addresses of the remote devices instead of their names. This option is available for debugging purposes.

→ *showLastList*
If true, causes all other parameters to be ignored and displays the same list as the previous call to `BtLibDiscoverMultipleDevices`.

Returns Returns one of the following values:

`btLibErrNoError`
Success

`btLibErrCanceled`
User canceled discovery.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments This blocking call performs a full discovery for an application, including name and feature retrieval and testing. This function takes over the UI and presents a choice box to the user, allowing the user to select multiple devices from the list of devices that were discovered. This function does not return until the user chooses one or more devices, or cancels.

Setting the `showLastList` parameter to `true` allows you to present to the user the list of devices displayed in the previous call to `BtLibDiscoverMultipleDevices` or `BtLibDiscoverSingleDevice()`. This feature can be useful because a full discovery process takes approximately ten seconds. The cached device list remains valid even after you close the library, allowing other Bluetooth applications to use it.

Note that BtLibStartInquiry() overwrites the cached device list. If you are using the `showLastList` feature, you should avoid calling `BtLibStartInquiry` between calls to `BtLibDiscoverMultipleDevices` or `BtLibDiscoverSingleDevice`.

Use BtLibGetSelectedDevices() to retrieve the list of devices that the user selected.

See Also BtLibGetSelectedDevices(), BtLibDiscoverSingleDevice()

BtLibDiscoverSingleDevice Function

Purpose Discover all available devices, present them in the user interface, and allow the user to select one of these devices.

Declared In `BtLib.h`

Prototype

```
Err BtLibDiscoverSingleDevice
    (UInt16 btLibRefNum, Char *instructionTxt,
    BtLibClassOfDeviceType *deviceFilterList,
    UInt8 deviceFilterListLen,
    BtLibDeviceAddressType *selectedDeviceP,
    Boolean addressAsName, Boolean showLastList)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *instructionTxt*
Text displayed at the top of the selection box. Pass `NULL` to display the default text. The default text is "Select a device:"

→ *deviceFilterList*
Array of BtLibClassOfDeviceTypes. This function displays only the remote devices whose class matches a class in this list. If `deviceFilterList` is `NULL`, this function displays all discovered devices.

→ *deviceFilterListLen*
Number of elements in `deviceFilterList`.

← *selectedDeviceP*
Pointer to a BtLibDeviceAddressType where this function stores the address of the device the user selects. You need to allocate this space before calling this function.

→ *addressAsName*
If `true`, display the Bluetooth addresses of the remote devices instead of their names. This option is available for debugging purposes.

→ *showLastList*
If `true`, causes all other parameters to be ignored and displays the same list as the previous call to `BtLibDiscoverSingleDevice`.

Returns Returns one of the following values:

`btLibErrNoError`
Success

`btLibErrCanceled`
User canceled the discovery.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments This blocking call performs a full discovery for an application, including name and feature retrieval and testing. This function takes over the UI and presents a choice box to the user, allowing the user to select a device from the list of devices that were discovered. This function does not return until the user chooses a device or cancels.

Setting the `showLastList` parameter to `true` allows you to present to the user the list of devices displayed in the previous call to `BtLibDiscoverSingleDevice` or `BtLibDiscoverMultipleDevices()`. This feature can be useful because a full discovery process takes approximately ten seconds. The cached device list remains valid even after you close the library, allowing other Bluetooth applications to use it.

Note that `BtLibStartInquiry()` overwrites the cached device list. If you are using the `showLastList` feature, you should avoid calling `BtLibStartInquiry` between calls to `BtLibDiscoverSingleDevice` or `BtLibDiscoverMultipleDevices`.

See Also `BtLibDiscoverMultipleDevices()`

BtLibGetGeneralPreference Function

Purpose Get one of the general management preferences.

Declared In `BtLib.h`

Prototype

```
Err BtLibGetGeneralPreference
    (UInt16 btLibRefNum,
    BtLibGeneralPrefEnum pref, void *prefValue,
    UInt16 prefValueSize)
```

Parameters

→ *btLibRefNum*
: Reference number for the Bluetooth library.

→ *pref*
: General preference to get.

← *prefValue*
: Pointer to a buffer to hold the value of the preference. You must allocate this buffer. This parameter must not be `NULL`.

→ *prefValueSize*
: Size, in bytes, of the `prefValue` buffer. You must set this size so it matches the size of the retrieved preference.

Returns Returns one of the following values:

`btLibErrNoError`
: Success.

`btLibErrNotOpen`
: The referenced Bluetooth library is not open.

`btLibErrParamError`
: One or more parameters is invalid. Be sure that the `prefValueSize` parameter matches the size of the preference value.

`btLibErrStackNotOpen`
: The Bluetooth stack failed to initialize when the library was opened.

Comments Specify the preference with a member of the `BtLibGeneralPreferenceEnum`.

BtLibGeneralPreferenceEnum

The `BtLibGeneralPreferenceEnum` enum specifies the general preferences that can be accessed using the

BtLibSetGeneralPreference() and BtLibGetGeneralPreference() functions.

Prototype

```
typedef enum {
  btLibPref_Name,
  btLibPref_UnconnectedAccessible,
  btLibPref_CurrentAccessible,
  btLibPref_LocalClassOfDevice,
  btLibPref_LocalDeviceAddress
} BtLibGeneralPrefEnum;
```

Constants

`btLibPref_CurrentAccessible`
: This preference is a BtLibAccessibleModeEnum indicating the current accessibility mode of the local device.

`btLibPref_LocalClassOfDevice`
: This preference is a BtLibClassOfDeviceType indicating the class of the local device. You should never set this preference.

`btLibPref_Name`
: This preference is a BtLibFriendlyNameType containing the user-friendly name of the local device. If you retrieve this preference, you also need to allocate a buffer and set the `BtLibFriendlyNameType`'s `name` and `nameLength` fields to the buffer pointer and buffer length, respectively.

  You should never set this preference.

`btLibPref_UnconnectedAccessible`
: This preference is a BtLibAccessibleModeEnum indicating the accessibility mode of the local device when it is unconnected. You should never set this preference.

See Also

BtLibSetGeneralPreference()

BtLibGetRemoteDeviceName Function

Purpose Get the name of the remote device with the specified address.

Declared In `BtLib.h`

Prototype

```
Err BtLibGetRemoteDeviceName (UInt16 btLibRefNum,
    BtLibDeviceAddressTypePtr remoteDeviceP,
    BtLibFriendlyNameType *nameP,
    BtLibGetNameEnum retrievalMethod)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *remoteDeviceP*
Pointer to a `BtLibDeviceAddressType` containing the address of the device whose name is desired.

↔ *nameP*
Pointer to a `BtLibFriendlyNameType` structure in which to store the results of the lookup. You must allocate this structure and the name buffer it points to. You also must specify the size of the buffer in the `nameLength` field of the structure. When the function returns, ignore the `nameLength` field and use `StrLen` to obtain the actual length of the name. This parameter must not be `NULL`.

→ *retrievalMethod*
Method used to retrieve the user-friendly remote device name. See `BtLibGetNameEnum`.

Returns Returns one of the following values:

`btLibErrNoError`
The name structure was successfully retrieved from the cache. No notification event will be generated.

`btLibErrBusy`
There is already a name request pending.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrPending`
The results will be returned through a notification.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments

The Bluetooth library maintains a cache of 50 device names. If the `retrievalMethod` parameter is `btLibCachedThenRemote`, this function first checks the cache for a name. If the name is in the cache, the value is returned immediately in the `nameP` parameter. If the name is not in the cache, the function queries the remote device for its name, forming a temporary ACL connection if one is not already in place. In this case, the function returns `btLibErrPending` and generates a `btLibManagementEventNameResult` event when the name is available.

Other values of the `retrievalMethod` parameter can instruct this function to look for the name only in the cache or only on the remote device. See `BtLibGetNameEnum` for more information.

BtLibGetNameEnum

The `BtLibGetNameEnum` enum specifies whether to retrieve a device name from the cache, the remote device, or both.

Prototype

```
typedef enum {
  btLibCachedThenRemote,
  btLibCachedOnly,
  btLibRemoteOnly
} BtLibGetNameEnum;
```

Constants

`btLibCachedOnly`
: Look for a name in the cache. If the name is not in the cache, fail.

`btLibCachedThenRemote`
: Look for a name in the cache. If the name is not in the cache, ask the remote device.

`btLibRemoteOnly`
: Ignore any cached names and ask the remote device for its name.

BtLibGetSelectedDevices Function

Purpose Get the list of devices selected during the last call to `BtLibDiscoverMultipleDevices()`.

Declared In `BtLib.h`

Prototype

```
Err BtLibGetSelectedDevices (UInt16 btLibRefNum,
   BtLibDeviceAddressType *selectedDeviceArray,
   UInt8 arraySize, UInt8 *numDevicesReturned)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

`selectedDeviceArray`
Array into which the results of the `BtLibDiscoverMultipleDevices()` function should be placed. You must allocate this array of `BtLibDeviceAddressType`s.

→ *arraySize*
Number of elements in the `selectedDeviceArray` you allocated.

← *numDevicesReturned*
Number of results placed in `selectedDeviceArray`.

Returns Returns `btLibErrNoError` if the query is successful. Returns `btLibErrNotOpen` if the referenced Bluetooth library is not open or `btLibErrStackNotOpen` if the Bluetooth stack failed to initialize when the library was opened.

Comments No callback events.

See Also `BtLibDiscoverMultipleDevices()`

BtLibLinkConnect Function

Purpose Create a Bluetooth Asynchronous Connectionless (ACL) link.

Declared In `BtLib.h`

Prototype

```
Err BtLibLinkConnect (UInt16 btLibRefNum,
   BtLibDeviceAddressTypePtr remoteDeviceP)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *remoteDeviceP*
Pointer to the a BtLibDeviceAddressType containing the address of the remote device.

Returns Returns one of the following values:

btLibErrPending
The results will be returned through a callback event.

btLibErrAlreadyConnected
The device is already in a pre-existing connection and cannot create a new connection.

btLibErrBluetoothOff
The Bluetooth radio is off. The user can turn the radio on and off with a setting in the preferences panel.

btLibErrBusy
A piconet is currently being created or destroyed.

btlibErrNotOpen
The referenced Bluetooth library is not open.

btLibErrStackNotOpen
The Bluetooth stack failed to initialize when the library was opened.

btLibErrTooMany
Cannot create another ACL link because the maximum allowed number has already been reached.

Comments An ACL link is a packet-switched physical level connection between two devices that is needed before the devices can form a RFCOMM or L2CAP connection.

When the connection is established or if it fails to be established, the btLibManagementEventAclConnectOutbound event is generated.

See Also BtLibLinkDisconnect()

BtLibLinkDisconnect Function

Purpose Disconnect an existing ACL Link.

Declared In `BtLib.h`

Prototype

```
Err BtLibLinkDisconnect (UInt16 btLibRefNum,
    BtLibDeviceAddressTypePtr remoteDeviceP)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *remoteDeviceP*
Pointer to a `BtLibDeviceAddressType` containing the address of the remote device.

Returns Returns one of the following values:

`btLibErrNoError`
The connection attempt was canceled before it started. No event is generated.

`btLibErrPending`
When the link actually disconnects, a `btLibManagementEventAclDisconnect` callback event is generated.

`btLibErrBusy`
Can't disconnect the link because the piconet is being destroyed.

`btLibErrNoConnection`
No link to the specified device exists.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments When the link disconnects, a `btLibManagementEventAclDisconnect` event is generated.

See Also `BtLibLinkConnect()`

BtLibLinkGetState Function

Purpose Get the state of an ACL link.

Declared In `BtLib.h`

Prototype

```
Err BtLibLinkGetState(UInt16 btLibRefNum,
   BtLibDeviceAddressTypePtr remoteDeviceP,
   BtLibLinkPrefsEnum pref, void *linkState,
   UInt16 linkStateSize)
```

Parameters

→ *btLibRefNum*
: Reference number for the Bluetooth library.

→ *remoteDeviceP*
: Pointer to a `BtLibDeviceAddressType` containing the address of the remote device. This address identifies the ACL link.

→ *pref*
: Link preference to retrieve. See `BtLibLinkPrefsEnum`.

← *linkState*
: Pointer to a buffer to store the value of the preference. You must allocate this buffer. This parameter must not be `NULL`. See `BtLibLinkPrefsEnum` for more information.

→ *linkStateSize*
: Size, in bytes, of `linkState` buffer. This size must match the size of the retrieved preference.

Returns Returns one of the following values:

`btLibErrNoError`
: Success. The `linkState` variable has been filled in.

`btLibErrNoAclLink`
: No link to the specified remote device exists.

`btLibErrNotOpen`
: The referenced Bluetooth library is not open.

`btLibErrParamError`
: The `linkStateSize` parameter is not same as the size of the preference value.

`btLibErrStackNotOpen`
: The Bluetooth stack failed to initialize when the library was opened.

Comments See the `BtLibLinkSetState()` function description for a list of the link states preferences.

See Also `BtLibLinkSetState()`

BtLibLinkSetState Function

Purpose Set the state of an ACL link

Declared In `BtLib.h`

Prototype

```
Err BtLibLinkSetState (UInt16 btLibRefNum,
   BtLibDeviceAddressTypePtr remoteDeviceP,
   BtLibLinkPrefsEnum pref, void *linkState,
   UInt16 linkStateSize)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *remoteDeviceP*
The address of the remote device. This address identifies the ACL link.

→ *pref*
Link preference to set. See `BtLibLinkPrefsEnum`.

→ *linkState*
Pointer to the value of the preference. See `BtLibLinkPrefsEnum`.

→ *linkStateSize*
Size, in bytes, of the *linkState* value.

Returns Returns one of the following values:

`btLibErrPending`
The results will be returned through a callback event.

`btLibErrFailed`
An attempt was made to encrypt a link before authenticating it.

`btLibErrNoAclLink`
No link to the specified remote device exists.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrParamError`
: The preference cannot be set or *`linkStateSize`* is invalid.

`btLibErrStackNotOpen`
: The Bluetooth stack failed to initialize when the library was opened.

Comments

Applications use this function to set the state of an ACL link. This function may generate events depending on the preference you change. The `btLibManagementEventAuthentication Complete` event indicates the link authentication has completed. The `btLibManagementEventEncryptionChange` indicates that the encryption has been enabled or disabled.

BtLibLinkPrefsEnum

The `BtLibLinkPrefsEnum` enum specifies the link state preferences that can be accessed with the `BtLibLinkSetState()` and `BtLibLinkGetState()` functions.

Prototype

```
typedef enum {
  btLibLinkPref_Authenticated,
  btLibLinkPref_Encrypted,
  btLibLinkPref_LinkRole
} BtLibLinkPrefsEnum;
```

Constants

`btLibLinkPref_Authenticated`
: This preference is a `Boolean` and indicates whether the link has been authenticated or not. The *`linkState`* and *`linkStateSize`* parameters are ignored.

`btLibLinkPref_Encrypted`
: This preference is a `Boolean` and indicates whether the link is encrypted or not.

`btLibLinkPref_LinkRole`
: This preference is a `BtLibConnectionRoleEnum` and indicates whether the remote device is a master or a slave. You cannot set this preference but you can get its value.

See Also

`BtLibLinkGetState()`

BtLibPiconetCreate Function

Purpose Create a piconet or reconfigure an existing piconet so the local device is the master.

Declared In `BtLib.h`

Prototype

```
Err BtLibPiconetCreate (UInt16 btLibRefNum,
   Boolean unlockInbound, Boolean discoverable)
```

Parameters

→ *`btLibRefNum`*
Reference number for the Bluetooth library.

→ *`unlockInbound`*
If `true`, the piconet accepts inbound connections. Otherwise, the piconet only allows outbound connections.

→ *`discoverable`*
If `true`, configures the radio to be discoverable. In other words, the radio responds to inquiries. If `false`, configures the radio to be only connectable. In other words, only devices that know the radio's Bluetooth device address can connect to it. This parameter is ignored if `unlockInbound` is `false`.

Returns Returns one of the following values:

`btLibErrNoError`
Successfully created the piconet with the local device as the master. No callback event is generated.

`btLibErrPending`
An existing ACL link needs to switch roles before this operation can complete.

`btLibErrFailed`
A piconet already exists.

`btLibErrInProgress`
Another piconet is currently being created.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments If the accessibility of the radio changes due to this operation, a `btLibManagementEventAccessibilityChange` event is

generated. When the piconet is created, or if the piconet fails to be created, a btLibManagementEventPiconetCreated event is generated. The status field of the BtLibManagementEventType structure accompanying the event indicates whether the piconet was created or not.

See Also BtLibPiconetDestroy(), BtLibPiconetUnlockInbound(), BtLibPiconetLockInbound()

BtLibPiconetDestroy Function

Purpose Destroy the piconet by disconnecting links to all devices and removing all restrictions on whether the local device is a master or a slave.

Declared In BtLib.h

Prototype

```
Err BtLibPiconetDestroy (UInt16 btLibRefNum)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

Returns Returns one of the following values:

btLibErrNoError
Successfully destroyed the piconet. A btLibManagementEventPiconetDestroyed event is not generated.

btLibErrPending
The piconet is being destroyed, and a btLibManagementEventPiconetDestroyed event will be generated when the operation succeeds or fails.

btLibErrBusy
The piconet is already in the process of being destroyed.

btLibErrNoPiconet
No piconet exists to be destroyed.

btLibErrNotOpen
The referenced Bluetooth library is not open.

btLibErrStackNotOpen
The Bluetooth stack failed to initialize when the library was opened.

Comments A `btLibManagementEventAclDisconnect` event is generated for each ACL link that is disconnected. When the piconet is successfully destroyed or fails to be destroyed, a `btLibManagementEventPiconetDestroyed` is generated. The `status` field of the `BtLibManagementEventType` structure accompanying the event indicates whether the piconet was destroyed or not.

See Also `BtLibPiconetCreate()`

BtLibPiconetLockInbound Function

Purpose Prevent remote devices from creating ACL links into the piconet.

Declared In `BtLib.h`

Prototype

```
Err BtLibPiconetLockInbound (UInt16 btLibRefNum)
```

Parameters

→ *`btLibRefNum`*
Reference number for the Bluetooth library.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrBusy`
The piconet is in the process of being destroyed.

`btLibErrNoPiconet`
No piconet exists.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments After locking inbound connections, outbound connections are still allowed. Locking inbound connections maximizes the bandwidth for members of the piconet to transmit data to each other.

See Also `BtLibPiconetUnlockInbound()`

BtLibPiconetUnlockInbound Function

Purpose Allow remote devices to create ACL links into the piconet.

Declared In `BtLib.h`

Prototype

```
Err BtLibPiconetUnlockInbound
    (UInt16 btLibRefNum, Boolean discoverable)
```

Parameters

→ *`btLibRefNum`*
Reference number for the Bluetooth library.

→ *`discoverable`*
If `true`, configures the radio to be discoverable. In other words, the radio responds to inquiries. If `false`, configures the radio to be only connectable. In other words, only devices that know the radio's Bluetooth device address can connect to it.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrBusy`
The piconet is in the process of being destroyed.

`btLibErrNoPiconet`
No piconet exists.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments Allowing inbound connections lowers the bandwidth available to transmit data between members of the piconet because the radio must periodically scan for incoming links.

See Also `BtLibPiconetLockInbound()`

BtLibRegisterManagementNotification Function

Purpose Register a callback function to process events generated by management functions.

Declared In `BtLib.h`

Prototype

```
Err BtLibRegisterManagementNotification
    (UInt16 btLibRefNum,
    BtLibManagementProcPtr callbackP,
    UInt32 refCon)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *callbackP*
Pointer to a callback procedure to register. This pointer must not be `NULL`. See `BtLibManagementCallback()` for more information.

→ *refCon*
Application-defined data to pass to the event handler.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrAlreadyRegistered`
The callback has already been registered with the management entity.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrParamError`
One or more parameters is invalid.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

`btLibErrTooMany`
There is no space available to store the callback. The management entity allows a maximum of 16 callbacks to be registered at a time.

Comments The management functions are asynchronous. That is, they return immediately and generate events when the task actually completes at a later time. To handle these events, you need to define a callback function with the same prototype as `BtLibManagementCallback()`. Then you need to register your callback function using `BtLibRegisterManagementNotification`. For examples of the callback events your callback function needs to handle, see the Management Callback Events section.

Applications should unregister their management callbacks before closing the Bluetooth library to prevent the callback table from overflowing. The callback table holds a maximum of 16 entries.

See Also `BtLibUnregisterManagementNotification()`

BtLibSetGeneralPreference Function

Purpose Set one of the general management preferences.

Declared In `BtLib.h`

Prototype

```
Err BtLibSetGeneralPreference
    (UInt16 btLibRefNum,
    BtLibGeneralPrefEnum pref, void *prefValue,
    UInt16 prefValueSize)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *pref*
General preference to set. See `BtLibGeneralPreferenceEnum`.

→ *prefValue*
Pointer to the value of the preference. This parameter must not be `NULL`. See `BtLibGeneralPreferenceEnum`.

→ *prefValueSize*
Size, in bytes, of `prefValue`.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrPending`
The results will be returned through a notification.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrParamError`
One or more parameters is invalid. Be sure that `prefValueSize` matches the size of the preference value.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments See the `BtLibGetGeneralPreference()` function description for a list of the preferences.

This function may generate events depending on the preference you change. The `btLibManagementEventAccessibilityChange` event indicates that the accessibility of the local device has changed.

See Also `BtLibGetGeneralPreference()`

BtLibStartInquiry Function

Purpose Start a Bluetooth inquiry.

Declared In `BtLib.h`

Prototype

```
Err BtLibStartInquiry (UInt16 btLibRefNum,
    UInt8 timeout, UInt8 maxResp)
```

Parameters

→ *`btLibRefNum`*
Reference number for the Bluetooth library.

→ *`timeout`*
Time, in seconds, this inquiry is allowed to take. If the inquiry does not complete within this time, it is canceled. The actual time is rounded to the nearest multiple of 1.28 seconds. If you specify a timeout period larger than 60 seconds, this function acts as if you specified a timeout period of 60 seconds. If this parameter is 0, the timeout period defaults to 10.24 seconds as specified in the Generic Access Profile.

→ *`maxResp`*
Maximum number of responses the inquiry accepts. Responses are not guaranteed to be unique.

Returns Returns one of the following values:

`btLibErrPending`
The results will be returned through callback events.

`btLibErrBluetoothOff`
The Bluetooth radio is off. The user can turn the radio on and off with a setting in the preferences panel.

`btLibErrInProgress`
Another inquiry is already in progress.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments The function performs a low-level Bluetooth inquiry, as opposed to a full device discovery. Specifically, inquiries started with this function only return the Bluetooth address and the class of the discovered device. This function does not have a user interface.

Every time a device is discovered, a `btLibManagementEventInquiryResult` callback event is generated. When the inquiry is complete, a `btLibManagementEventInquiryComplete` callback event is generated. If the application calls `BtLibCancelInquiry()`, a `btLibManagementEventInquiryCanceled` callback event is generated.

See Also `BtLibCancelInquiry()`

BtLibUnregisterManagementNotification Function

Purpose Unregister a previously registered management callback.

Declared In `BtLib.h`

Prototype

```
Err BtLibUnregisterManagementNotification
    (UInt16 btLibRefNum,
    BtLibManagementProcPtr callbackP)
```

Parameters

→ *`btLibRefNum`*
Reference number for the Bluetooth library.

→ *`callbackP`*
Pointer to the callback procedure to unregister. This pointer must not be `NULL`.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrError`
The callback referenced by `callbackP` has not been registered.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrParamError`
One or more parameters are invalid.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments Applications should unregister their management callbacks before closing the library to prevent the callback table from overflowing. The callback table holds a maximum of 16 entries.

See Also `BtLibRegisterManagementNotification()`

Application-Defined Functions

This section describes the callback functions that handle management events. These functions are supplied by the developer and can be named anything.

BtLibManagementCallback Function

Purpose Signal the result of a Bluetooth management event. When the event takes place, this callback function is called.

Declared In `BtLibTypes.h`

Prototype

```
void (*BtLibManagementProcPtr)
    (BtLibManagementEventType *mEvent,
    UInt32 refCon)
```

Parameters

→ *mEvent*
: `BtLibManagementEventType` structure containing the event parameters.

→ *refCon*
: General purpose integer which you can use to hold application-specific information. When you register the callback with the `BtLibRegisterManagementNotification()` function, you can specify a value to pass to this parameter.

Returns Returns nothing.

Comments The event and status of the event are in the `mEvent` structure. See Management Callback Events for more information.

You must register this function using the `BtLibRegisterManagementNotification()` function before it starts receiving events.

83

Bluetooth Library: Sockets and Service Discovery

The Bluetooth library uses sockets to represent L2CAP, RFCOMM, and SDP connections. This chapter presents reference material for the socket and SDP support provided by the Bluetooth library API:

`Socket-Related Data Structures`
: This section lists some of the more important types used by the Bluetooth library.

`Socket Callback Events`
: The Bluetooth library socket API supports L2CAP, RFCOMM, and SDP, the upper protocols of the Bluetooth specification. As with the management functions, the socket functions are mostly asynchronous. To signal to the application that socket operations have completed, the Bluetooth library generates **socket callback events** by calling a callback function.

`Socket Disconnection Error Codes`
: In addition to the standard error codes that can accompany socket events, the status codes accompanying the disconnection and connection events can have the additional values enumerated in this section.

`Socket Functions`
: The functions in this section perform general socket tasks and tasks related to L2CAP and RFCOMM sockets. The functions specific to SDP sockets are in the next section.

`Service Discovery Protocol Functions`
: This section describes functions and macros specific to the Bluetooth Service Discovery Protocol (SDP).

Application-Defined Functions
: This section describes the callback functions that handle socket events.

The header file `BtLib.h` declares the Bluetooth library functions and macros. The header file `BtLibTypes.h` declares the data structures that you use with those functions and macros.

Socket-Related Data Structures

This section lists some of the more important types used by the Bluetooth library.

BtLibL2CapPsmType Typedef

Purpose The `BtLibL2CapPsmType` type represents a Protocol and Server Multiplexer (PSM) value. See the "Logical Link and Adaptation Protocol Specification" chapter of the *Specification of the Bluetooth System* for more information about PSM values. The Bluetooth library only supports two-byte PSM values.

Prototype

```
typedef UInt16 BtLibL2CapPsmType;
```

BtLibLanguageBaseTripletType Struct

Purpose The `BtLibLanguageBaseTripletType` structure represents a language base attribute identifier list attribute. See the "Service Discovery Protocol" chapter of the *Specification of the Bluetooth System* for more information.

Prototype

```
typedef struct BtLibLanguageBaseTripletType {
  UInt16 naturalLanguageIdentifier;
  UInt16 characterEncoding;
  UInt16 baseAttributeID;
} BtLibLanguageBaseTripletType;
```

Fields

`naturalLanguageIdentifier`
: A `UInt16` representing a natural language. See Language Constants for a set of constants that can be used in this field.

`characterEncoding`
: A `UInt16` representing a character set encoding. See Character Encoding Constants for a set of constants that can be used in this field.

`baseAttributeID`
: Base attribute identifiers for attributes represented in this language. See Attribute Identifier Offsets for offsets that are added to this value to get the attribute identifiers for specific attributes represented in this language.

Language Constants

These constants are used in the `naturalLanguageIdentifier` field of the `BtLibLanguageBaseTripletType` and are defined in the ISO 639:1988 specification.

Constants

`btLibLangAfar`

`btLibLangAbkihazian`

`btLibLangAfrikaans`

`btLibLangAmharic`

`btLibLangArabic`

`btLibLangAssamese`

`btLibLangAymara`

`btLibLangAzerbaijani`

`btLibLangBashkir`

`btLibLangByelorussian`

`btLibLangBulgarian`

`btLibLangBihari`

`btLibLangBislama`

`btLibLangBengali`

`btLibLangTibetan`

`btLibLangBreton`

`btLibLangCatalan`

`btLibLangCorsican`

```
btLibLangCzech
btLibLangWelsh
btLibLangDanish
btLibLangGerman
btLibLangBhutani
btLibLangGreek
btLibLangEnglish
btLibLangEsperanto
btLibLangSpanish
btLibLangEstonian
btLibLangBasque
btLibLangPersian
btLibLangFinnish
btLibLangFiji
btLibLangFaroese
btLibLangFrench
btLibLangFrisian
btLibLangIrish
btLibLangScotsGaelic
btLibLangGalician
btLibLangGuarani
btLibLangGujarati
btLibLangHausa
btLibLangHindi
btLibLangCroation
btLibLangHungarian
btLibLangArmenian
btLibLangInterlingua
```

```
btLibLangInterlingue
btLibLangInupiak
btLibLangIndonesian
btLibLangIcelandic
btLibLangItalian
btLibLangHebrew
btLibLangJapanese
btLibLangYiddish
btLibLangJavanese
btLibLangGeorgian
btLibLangKazakh
btLibLangGreenlandic
btLibLangCambodian
btLibLangKannada
btLibLangKorean
btLibLangKashmiri
btLibLangKurdish
btLibLangKirghiz
btLibLangLatin
btLibLangLingala
btLibLangLaothian
btLibLangLithuanian
btLibLangLatvian
btLibLangMalagasy
btLibLangMaori
btLibLangMacedonian
btLibLangMalayalam
btLibLangMongolian
```

```
btLibLangMoldavian
btLibLangMarathi
btLibLangMalay
btLibLangMaltese
btLibLangBurmese
btLibLangNaura
btLibLangNepali
btLibLangDutch
btLibLangNorwegian
btLibLangOccitan
btLibLangOromo
btLibLangOriya
btLibLangPunjabi
btLibLangPolish
btLibLangPashto
btLibLangPortuguese
btLibLangQuechua
btLibLangRhaeto_Romance
btLibLangKirundi
btLibLangRomanian
btLibLangRussian
btLibLangKinyarwanda
btLibLangSanskrit
btLibLangSindhi
btLibLangSangho
btLibLangSerbo_Croation
btLibLangSinghalese
btLibLangSlovak
```

```
btLibLangSlovenian
btLibLangSamoan
btLibLangShona
btLibLangSomali
btLibLangAlbanian
btLibLangSerbian
btLibLangSiswati
btLibLangSesotho
btLibLangSundanese
btLibLangSwedish
btLibLangSwahili
btLibLangTamil
btLibLangTelugu
btLibLangTajik
btLibLangThai
btLibLangTigrinya
btLibLangTurkmen
btLibLangTagalog
btLibLangSetswanna
btLibLangTonga
btLibLangTurkish
btLibLangTsonga
btLibLangTatar
btLibLangTwi
btLibLangUkranian
btLibLangUrdu
btLibLangUzbek
btLibLangVietnamese
```

`btLibLangVolapuk`

`btLibLangWolof`

`btLibLangXhosa`

`btLibLangYoruba`

`btLibLangChinese`

`btLibLangZulu`

Character Encoding Constants

These constants are used to specify the character encoding used for SDP attributes. More information about these character sets can be found at `http://www.iana.org/assignments/character-sets`.

Constants

`btLibCharSet_US_ASCII`

`btLibCharSet_Adobe_Standard_Encoding`

`btLibCharSet_Adobe_Symbol_Encoding`

`btLibCharSet_ANSI_X3_110_1983`

`btLibCharSet_ASMO_449`

`btLibCharSet_Big5`

`btLibCharSet_Big5_HKSCS`

`btLibCharSet_BS_4730`

`btLibCharSet_BS_viewdata`

`btLibCharSet_CSA_Z243_4_1985_1`

`btLibCharSet_CSA_Z243_4_1985_2`

`btLibCharSet_CSA_Z243_4_1985_gr`

`btLibCharSet_CSN_369103`

`btLibCharSet_DEC_MCS`

`btLibCharSet_DIN_66003`

`btLibCharSet_dk_us`

`btLibCharSet_DS_2089`

`btLibCharSet_EBCDIC_AT_DE`

```
btLibCharSet_EBCDIC_AT_DE_A
btLibCharSet_EBCDIC_CA_FR
btLibCharSet_EBCDIC_DK_NO
btLibCharSet_EBCDIC_DK_NO_A
btLibCharSet_EBCDIC_ES
btLibCharSet_EBCDIC_ES_A
btLibCharSet_EBCDIC_ES_S
btLibCharSet_EBCDIC_FI_SE
btLibCharSet_EBCDIC_FI_SE_A
btLibCharSet_EBCDIC_FR
btLibCharSet_EBCDIC_IT
btLibCharSet_EBCDIC_PT
btLibCharSet_EBCDIC_UK
btLibCharSet_EBCDIC_US
btLibCharSet_ECMA_cyrillic
btLibCharSet_ES 23
btLibCharSet_ES2
btLibCharSet_EUC_JP
btLibCharSet_EUC_KR
btLibCharSet_Extended_UNIX_Code_Fixed_Width_for_Ja
  panese
btLibCharSet_GB2312
btLibCharSet_GB_1988_80
btLibCharSet_GB_2312_80
btLibCharSet_GOST_19768_74
btLibCharSet_greek7
btLibCharSet_greek7_old
btLibCharSet_greek_ccitt
btLibCharSet_HP_DeskTop
```

```
btLibCharSet_HP_Legal
btLibCharSet_HP_Math8
btLibCharSet_HP_Pi_font
btLibCharSet_hp_roman8
btLibCharSet_HZ_GB_2312
btLibCharSet_IBM037
btLibCharSet_IBM038
btLibCharSet_IBM273
btLibCharSet_IBM274
btLibCharSet_IBM275
btLibCharSet_IBM277
btLibCharSet_IBM278
btLibCharSet_IBM280
btLibCharSet_IBM281
btLibCharSet_IBM284
btLibCharSet_IBM285
btLibCharSet_IBM290
btLibCharSet_IBM297
btLibCharSet_IBM420
btLibCharSet_IBM423
btLibCharSet_IBM424
btLibCharSet_IBM437
btLibCharSet_IBM500
btLibCharSet_IBM775
btLibCharSet_IBM850
btLibCharSet_IBM851
btLibCharSet_IBM852
btLibCharSet_IBM855
```

```
btLibCharSet_IBM857
btLibCharSet_IBM860
btLibCharSet_IBM861
btLibCharSet_IBM862
btLibCharSet_IBM863
btLibCharSet_IBM864
btLibCharSet_IBM865
btLibCharSet_IBM866
btLibCharSet_IBM868
btLibCharSet_IBM869
btLibCharSet_IBM870
btLibCharSet_IBM871
btLibCharSet_IBM880
btLibCharSet_IBM891
btLibCharSet_IBM903
btLibCharSet_IBM904
btLibCharSet_IBM905
btLibCharSet_IBM918
btLibCharSet_IBM1026
btLibCharSet_IBM00858
btLibCharSet_IBM00924
btLibCharSet_IBM01140
btLibCharSet_IBM01141
btLibCharSet_IBM01142
btLibCharSet_IBM01143
btLibCharSet_IBM01144
btLibCharSet_IBM01145
btLibCharSet_IBM01146
```

```
btLibCharSet_IBM01147
btLibCharSet_IBM01148
btLibCharSet_IBM01149
btLibCharSet_IBM_Symbols
btLibCharSet_IBM_Thai
btLibCharSet_IEC_P27_1
btLibCharSet_INIS
btLibCharSet_INIS_8
btLibCharSet_INIS_cyrillic
btLibCharSet_INVARIANT
btLibCharSet_ISO_646_basic_1983
btLibCharSet_ISO_646_irv_1983
btLibCharSet_ISO_2022_CN
btLibCharSet_ISO_2022_CN_EXT
btLibCharSet_ISO_2022_JP
btLibCharSet_ISO_2022_JP_2
btLibCharSet_ISO_2022_KR
btLibCharSet_ISO_2033_1983
btLibCharSet_ISO_5427
btLibCharSet_ISO_5427_1981
btLibCharSet_ISO_5428_1980
btLibCharSet_ISO_6937_2_25
btLibCharSet_ISO_6937_2_add
btLibCharSet_ISO_8859_1
btLibCharSet_ISO_8859_10
btLibCharSet_iso_8859_13
btLibCharSet_iso_8859_14
btLibCharSet_ISO_8859_15
```

```
btLibCharSet_ISO_8859_1_Windows_3_0_Latin_1
btLibCharSet_ISO_8859_1_Windows_3_1_Latin_1
btLibCharSet_ISO_8859_2
btLibCharSet_ISO_8859_2_Windows_Latin_2
btLibCharSet_ISO_8859_3
btLibCharSet_ISO_8859_4
btLibCharSet_ISO_8859_5
btLibCharSet_ISO_8859_6
btLibCharSet_ISO_8859_6_E
btLibCharSet_ISO_8859_6_I
btLibCharSet_ISO_8859_7
btLibCharSet_ISO_8859_8
btLibCharSet_ISO_8859_8_E
btLibCharSet_ISO_8859_8_I
btLibCharSet_ISO_8859_9
btLibCharSet_ISO_8859_9_Windows_Latin_5
btLibCharSet_ISO_8859_supp
btLibCharSet_ISO_10367_box
btLibCharSet_ISO_10646_UCS_2
btLibCharSet_ISO_10646_UCS_4
btLibCharSet_ISO_10646_UCS_Basic
btLibCharSet_ISO_10646_Unicode_Latin1
btLibCharSet_ISO_10646_UTF_1
btLibCharSet_iso_ir_90
btLibCharSet_ISO_Unicode_IBM_1261
btLibCharSet_ISO_Unicode_IBM_1264
btLibCharSet_ISO_Unicode_IBM_1265
btLibCharSet_ISO_Unicode_IBM_1268
```

```
btLibCharSet_ISO_Unicode_IBM_1276
btLibCharSet_IT
btLibCharSet_JIS_C6220_1969_jp
btLibCharSet_JIS_C6220_1969_ro
btLibCharSet_JIS_C6226_1978
btLibCharSet_JIS_C6226_1983
btLibCharSet_JIS_C6229_1984_a
btLibCharSet_JIS_C6229_1984_b
btLibCharSet_JIS_C6229_1984_b_add
btLibCharSet_JIS_C6229_1984_hand
btLibCharSet_JIS_C6229_1984_hand_add
btLibCharSet_JIS_C6229_1984_kana
btLibCharSet_JIS_Encoding
btLibCharSet_JIS_X0201
btLibCharSet_JIS_X0212_1990
btLibCharSet_JUS_I_B1_002
btLibCharSet_JUS_I_B1_003_mac
btLibCharSet_JUS_I_B1_003_serb
btLibCharSet_KOI8_R
btLibCharSet_KOI8_U
btLibCharSet_KSC5636
btLibCharSet_KS_C_5601_1987
btLibCharSet_latin_greek
btLibCharSet_Latin_greek_1
btLibCharSet_latin_lap
btLibCharSet_macintosh
btLibCharSet_Microsoft_Publishing
btLibCharSet_MNEM
```

```
btLibCharSet_MNEMONIC
btLibCharSet_MSZ_7795_3
btLibCharSet_NATS_DANO
btLibCharSet_NATS_DANO_ADD
btLibCharSet_NATS_SEFI
btLibCharSet_NATS_SEFI_ADD
btLibCharSet_NC_NC00_10_81
btLibCharSet_NF_Z_62_010
btLibCharSet_NF_Z_62_010__1973_
btLibCharSet_NS_4551_1
btLibCharSet_NS_4551_2
btLibCharSet_PC8_Danish_Norwegian
btLibCharSet_PC8_Turkish
btLibCharSet_PT
btLibCharSet_PT2
btLibCharSet_SCSU
btLibCharSet_SEN_850200_B
btLibCharSet_SEN_850200_C
btLibCharSet_Shift_JIS
btLibCharSet_TIS_620
btLibCharSet_T_101_G2
btLibCharSet_T_61_7bit
btLibCharSet_T_61_8bit
btLibCharSet_UNICODE_1_1
btLibCharSet_UNICODE_1_1_UTF_7
btLibCharSet_UNKNOWN_8BIT
btLibCharSet_us_dk
btLibCharSet_UTF_16
```

`btLibCharSet_UTF_16BE`

`btLibCharSet_UTF_16LE`

`btLibCharSet_UTF_7`

`btLibCharSet_UTF_8`

`btLibCharSet_Ventura_International`

`btLibCharSet_Ventura_Math`

`btLibCharSet_Ventura_US`

`btLibCharSet_videotex_suppl`

`btLibCharSet_VIQR`

`btLibCharSet_VISCII`

`btLibCharSet_Windows_31J`

`btLibCharSet_windows_1250`

`btLibCharSet_windows_1251`

`btLibCharSet_windows_1252`

`btLibCharSet_windows_1253`

`btLibCharSet_windows_1254`

`btLibCharSet_windows_1255`

`btLibCharSet_windows_1256`

`btLibCharSet_windows_1257`

`btLibCharSet_windows_1258`

Attribute Identifier Offsets

Constants

`btLibServiceNameOffset`
: The offset from the `baseAttributeID` to get the attribute identifier of the service name.

`btLibServiceDescriptionOffset`
: The offset from the `baseAttributeID` to get the attribute identifier of the service description.

`btLibProviderNameOffset`
: The offset from the `baseAttributeID` to get the attribute identifier of the provider name.

BtLibProfileDescriptorListEntryType Struct

Purpose The `BtLibProfileDescriptorListEntryType` structure represents an entry in a profile descriptor list attribute. See the "Service Discovery Protocol" chapter of the *Specification of the Bluetooth System* for more information about profile descriptor list attributes.

Prototype

```
typedef struct
BtLibProfileDescriptorListEntryType {
  BtLibSdpUuidType profUUID;
  UInt16 version;
} BtLibProfileDescriptorListEntryType;
```

Fields

`profUUID`
: The UUID of the profile.

`version`
: The version of the profile.

BtLibProtocolDescriptorListEntryType Struct

Purpose The `BtLibProtocolDescriptorListEntryType` structure represents an entry in a protocol descriptor list attribute. See the "Service Discovery Protocol" chapter of the *Specification of the Bluetooth System* for more information.

Prototype

```
typedef struct
BtLibProtocolDescriptorListEntryType {
  BtLibSdpUuidType protoUUID;
  union {
    BtLibL2CapPsmType psm;
    BtLibRfCommServerIdType channel;
  } param;
} BtLibProtocolDescriptorListEntryType;
```

Fields

`protoUUID`
: The UUID of the protocol.

`param`
: A union containing two members: `psm` and `channel`. `psm` is applicable for a L2CAP protocol descriptor and specifies the Protocol and Service Multiplexor. `channel` is applicable to a RFCOMM protocol descriptor and specifies the server channel.

BtLibRfCommServerIdType Typedef

Purpose The `BtLibRfCommServerIdType` type represents a RFCOMM server channel. See the "RFCOMM with TS 07.10" chapter of the *Specification of the Bluetooth System* for more information about server channels.

Prototype

```
typedef UInt8 BtLibRfCommServerIdType;
```

BtLibSdpAttributeDataType Struct

Purpose The `BtLibSdpAttributeDataType` union is used to encapsulate an SDP attribute or a list entry in an SDP attribute. The `BtLibSdpServiceRecordGetAttribute()` function gets an attribute or a list entry and return its contents in a `BtLibSdpAttributeDataType`. The `BtLibSdpServiceRecordSetAttribute` function sets an attribute or list entry according to the contents of a `BtLibSdpAttributeDataType`. This type supports the universal attributes defined in the *Specification of the Bluetooth System*.

Prototype

```
typedef union BtLibSdpAttributeDataType {
  BtLibSdpUuidType serviceClassUuid;
  UInt32 serviceRecordState;
  BtLibSdpUuidType serviceIdUuid;
  BtLibProtocolDescriptorListEntryType
    protocolDescriptorListEntry;
  BtLibSdpUuidType browseGroupUuid;
  BtLibLanguageBaseTripletType
    languageBaseTripletListEntry;
  UInt32 timeToLive;
  UInt8 availability;
  BtLibProfileDescriptorListEntryType
    profileDescriptorListEntry;
  BtLibUrlType documentationUrl;
  BtLibUrlType clientExecutableUrl;
  BtLibUrlType iconUrl;
  BtLibStringType serviceName;
  BtLibStringType providerName;
} BtLibSdpAttributeDataType;
```

Note that if you're retrieving a string or a URL using the `BtLibSdpServiceRecordGetAttribute()` function, you first

need to allocate a buffer in addition to this union. This buffer must be large enough to contain the anticipated size of the string or URL. You must also initialize the string pointer and string length fields of the appropriate `BtLibAttributeDataType` union member. For example, if you're retrieving an icon URL, you need to set `iconURL.url` to point to the buffer. You also need to set `iconURL.urllen` to the length of the buffer.

See Also BtLibSdpUuidType, BtLibProtocolDescriptorListEntryType, BtLibProfileDescriptorListEntryType, BtLibLanguageBaseTripletType, BtLibUrlType, BtLibStringType

BtLibSdpAttributeIdType Typedef

Purpose The `BtLibSdpAttributeIdType` type represents a SDP attribute identifier.

Prototype

```
typedef UInt16 BtLibSdpAttributeIdType;
```

The following constants are defined by the Bluetooth library. They represent the universal attributes in the *Specification of the Bluetooth System*.

Constants

Universal Attribute IDs

`btLibServiceClassIdList`

`btLibServiceRecordState`

`btLibServiceId`

`btLibProtocolDescriptorList`

`btLibBrowseGroupList`

`btLibLanguageBaseAttributeIdList`

`btLibTimeToLive`

`btLibAvailability`

`btLibProfileDescriptorList`

`btLibDocumentationUrl`

`btLibClientExecutableUrl`

`btLibIconUrl`

BtLibSdpRecordHandle Typedef

Purpose The `BtLibSdpRecordHandle` type, also called an **SDP memory handle**, provides a memory handle to an **SDP memory record**.

Prototype

```
typedef MemHandle BtLibSdpRecordHandle;
```

A SDP memory record can have two roles: it can contain a local SDP service record or it can refer to an SDP service record on a remote device. In the latter role, the SDP memory record is said to be **mapped** to a service record on the remote device. The `BtLibSdpServiceRecordMapRemote()` function performs this mapping.

BtLibSdpRemoteServiceRecordHandle Typedef

Purpose The `BtLibSdpRemoteServiceRecordHandle` type represents a SDP service record handle on a remote device as defined in the "Service Discovery Protocol" chapter of the *Specification of the Bluetooth System*. The documentation refers to this type as a **remote service record handle**.

Prototype

```
typedef UInt32 BtLibSdpRemoteServiceRecordHandle;
```

Note that this type is different from the `BtLibSdpRecordHandle` type, which refers to a memory chunk containing an SDP service record.

BtLibSdpUuidSizeEnum Enum

Purpose The `BtLibSdpUuidSizeEnum` enum specifies the sizes that a UUID can have. See `BtLibSdpUuidType` for more information.

Prototype

```
typedef enum {
  btLibUuidSize16 = 2,
  btLibUuidSize32 = 4,
  btLibUuidSize128 = 16
} BtLibSdpUuidSizeEnum;
```

Constants

`btLibUuidSize16`
: 16-bit UUID

`btLibUuidSize32`
: 32-bit UUID

`btLibUuidSize128`
: Full size 128-bit UUID

BtLibSdpUuidType Struct

Purpose

The `BtLibSdpUuidType` structure represents a Universally Unique Identifier (UUID). A UUID is a 128-bit value that is generated in a manner that guarantees that it is different from every other UUID.

The "Service Discovery Protocol" chapter of the *Specification of the Bluetooth System* reserves a set of UUIDs for common Bluetooth services and protocols. You can specify these with 32 bits—the remaining 96 bits have a fixed value. A subset of these can be specified with 16 bits zero-extended to 32 bits. Therefore you can specify a UUID using 16, 32, or 128 bits.

You generally don't set this type directly. Instead, you use the `BtLibSdpUuidInitialize()` macro.

Prototype

```
typedef struct BtLibSdpUuidType {
  BtLibSdpUuidSizeEnum size;
  UInt8 UUID[16];
} BtLibSdpUuidType;
```

Fields

`size`
: The number of bits used to specify the UUID. See `BtLibSdpUuidSizeEnum`.

`UUID`
: The value of the UUID. If you're setting the value of this field, use the `BtLibSdpUuidInitialize()` macro.

Predefined UUIDs

Constants

`btLibSdpUUID_SC_SERVICE_DISCOVERY_SERVER`

`btLibSdpUUID_SC_BROWSE_GROUP_DESC`

`btLibSdpUUID_SC_PUBLIC_BROWSE_GROUP`

`btLibSdpUUID_SC_SERIAL_PORT`

```
btLibSdpUUID_SC_LAN_ACCESS_PPP
btLibSdpUUID_SC_DIALUP_NETWORKING
btLibSdpUUID_SC_IRMC_SYNC
btLibSdpUUID_SC_OBEX_OBJECT_PUSH
btLibSdpUUID_SC_OBEX_FILE_TRANSFER
btLibSdpUUID_SC_IRMC_SYNC_COMMAND
btLibSdpUUID_SC_HEADSET
btLibSdpUUID_SC_CORDLESS_TELEPHONY
btLibSdpUUID_SC_INTERCOM
btLibSdpUUID_SC_FAX
btLibSdpUUID_SC_HEADSET_AUDIO_GATEWAY
btLibSdpUUID_SC_WAP
btLibSdpUUID_SC_WAP_CLIENT
btLibSdpUUID_SC_PNP_INFO
btLibSdpUUID_SC_GENERIC_NETWORKING
btLibSdpUUID_SC_GENERIC_FILE_TRANSFER
btLibSdpUUID_SC_GENERIC_AUDIO
btLibSdpUUID_SC_GENERIC_TELEPHONY

btLibSdpUUID_PROT_SDP
btLibSdpUUID_PROT_RFCOMM
btLibSdpUUID_PROT_TCS_BIN
btLibSdpUUID_PROT_L2CAP
btLibSdpUUID_PROT_IP
btLibSdpUUID_PROT_UDP
btLibSdpUUID_PROT_TCP
btLibSdpUUID_PROT_TCS_AT
btLibSdpUUID_PROT_OBEX
```

`btLibSdpUUID_PROT_FTP`

`btLibSdpUUID_PROT_HTTP`

`btLibSdpUUID_PROT_WSP`

BtLibSocketEventType Struct

Purpose The `BtLibSocketEventType` structure contains detailed information regarding a socket callback event. All socket events have some common data. Most socket events have additional data specific to those events. The specific data is stored in a union that is part of the `BtLibSocketEvent` data structure.

Prototype

```
typedef struct _BtLibSocketEventType {
  BtLibSocketEventEnum event;
  BtLibSocketRef socket;
  Err status;
  union
  {
    ...
  } eventData;
} BtLibSocketEventType,
    *BtLibSocketEventTypePtr;
```

Fields

`event`
: `BtLibSocketEventEnum` enum member that indicates which socket event has occurred. See Socket Callback Events.

`socket`
: Socket associated with the event.

`status`
: Status of the event. The Socket Callback Events section gives more details about how to interpret this field for specific events.

`eventData`
: Data associated with the event. The member of this union that is valid depends on the event. See Socket Callback Events for more information.

BtLibSocketRef Typedef

Purpose The `BtLibSocketRef` type identifies a socket.

Prototype

```
typedef Int16 BtLibSocketRef;
```

BtLibStringType Struct

Purpose The `BtLibStringType` structure represents a string in an SDP attribute.

Prototype

```
typedef struct BtLibStringType {
  Char *str;
  UInt16 strLen;
} BtLibStringType;
```

Fields

`str`
: An array of characters representing the string. This array is not null-terminated.

`strLen`
: The length of the string.

BtLibUrlType Struct

Purpose The `BtLibUrlType` structure represents a uniform resource locator in an SDP attribute.

Prototype

```
typedef struct BtLibUrlType {
  Char *url;
  UInt16 urlLen;
} BtLibUrlType;
```

Fields

`url`
: An array of characters representing the URL. This array is not null-terminated.

`urlLen`
: The length of the string.

Socket Callback Events

The Bluetooth library socket API supports L2CAP, RFCOMM, and SDP, the upper protocols of the Bluetooth specification. As with the management functions, the socket functions are mostly asynchronous. To signal to the application that socket operations have completed, the Bluetooth library generates a **socket callback events** by calling a callback function.

You specify the callback function when you create a socket. When an event occurs, the callback function is called with two parameters: a pointer to a `BtLibSocketEventType` structure and a pointer to a user-defined structure.

The `BtLibSocketEventType` structure contains an `event` field, which indicates the reason the callback is called, a `status` field, which indicates status information associated with the event, a `socket` field, which indicates the socket associated with the event, and a union of several structures. The member of the union that is valid depends on the event. The meaning of the events is described in the following sections.

btLibSocketEventConnectedInbound

A remote connection has been accepted because the application has called `BtLibSocketRespondToConnection()`.

For this event, the `eventData` field contains the following field:

```
BtLibSocketRef newSocket;
```

This `BtLibSocketRef` contains the reference to the new socket.

If the remote device requests a L2CAP connection, this event is sent to the L2CAP listener socket with a PSM that matches the PSM of the requested connection. The Bluetooth library creates a new socket that exchanges data with the remote device.

If the remote device requests an RFCOMM connection, this event is sent to the RFCOMM listener socket with a server channel that matches the server channel of the requested connection. The Bluetooth library converts the listener socket into socket that exchanges data with the remote device.

btLibSocketEventConnectedOutbound

An outbound connection initiated by a call to `BtLibSocketConnect()` has completed. The `status` field is `btLibErrNoError` if the connection has completed successfully. Otherwise, the `status` field indicates why the connection failed.

btLibSocketEventConnectRequest

A remote device has requested a connection.

You must respond to this event with a call to `BtLibSocketRespondToConnection()`.

For this event, the `eventData` field contains the following field:

```
BtLibDeviceAddressType requestingDevice;
```

This `BtLibDeviceAddressType` contains the address of the remote device requesting the connection.

If the remote device requests a L2CAP connection, this event is sent to the L2CAP listener socket with a PSM that matches the PSM of the request.

If the remote device requests an RFCOMM connection, this event is sent to the RFCOMM listener socket with a server channel that matches the server channel of the request.

To convert a socket into a listener socket use the `BtLibSocketListen()` function.

btLibSocketEventData

Purpose Data has been received on a socket.

For this event, the `eventData` field contains the following structure:

Prototype

```
struct {
  UInt16 dataLen;
  UInt8 *data;
} data;
```

Fields

`dataLen`
The size, in bytes, of the received data.

`data`
A pointer to the received data.

btLibSocketEventDisconnected

The connection has been lost or one of the devices has disconnected. The socket is now invalid. The `status` field indicates the reason for the disconnection.

btLibSocketEventSdpServiceRecordHandle

Purpose

A request for remote service records matching a list of service classes has completed. The application initiated this request by calling the `BtLibSdpServiceRecordsGetByServiceClass()` function.

If the `status` field is `btLibErrNoError`, the SDP operation completed successfully, and the `eventData` field contains valid information. Otherwise the SDP operation failed, and the `status` field indicates the reason for the failure.

For this event, the `eventData` field contains the following structure:

Prototype

```
struct {
  UInt16 numSrvRec;
  BtLibSdpRemoteServiceRecordHandle
    *serviceRecordList;
} sdpServiceRecordHandles;
```

Fields

`numSrvRec`
Number of remote service record handles in the returned array.

`serviceRecordList`
An array of `BtLibSdpRemoteServiceRecordHandle`s for the service records matching the service class list.

btLibSocketEventSdpGetAttribute

Purpose

An attribute request has completed. The application initiated this request by calling the `BtLibSdpServiceRecordGetAttribute()` function.

If the `status` field is `btLibErrNoError`, the operation completed successfully, and the `eventData` field contains valid data. Otherwise the operation failed, and the `status` field indicates the reason for the failure.

For this event, the `eventData` field contains the following structure:

Prototype

```
struct {
  BtLibSdpAttributeIdType attributeID;
  BtLibSdpRecordHandle recordH;
  union {
    ...
  } info;
} sdpAttribute;
```

Fields

`attributeID`
: The attribute identifier of the attribute.

`recordH`
: A handle identifying the SDP memory record from which the attribute is retrieved.

`info`
: A union containing information specific to the event. See The info Field.

The info Field

For this event, the `info` field contains the following structure:

Prototype

```
struct {
  BtLibSdpAttributeDataType *attributeValues;
  UInt16 listNumber;
  UInt16 listEntry;
} data;
```

Fields

`attributeValues`
: A `BtLibSdpAttributeDataType` containing the value of the attribute or list entry.

`listNumber`
: The index of the list in which this list entry appears or 0 if the attribute is not a protocol descriptor list. The index of the first list is 0.

`listEntry`
: The index of the list entry within the list or 0 if the attribute is not a list. The index of the first entry is 0.

btLibSocketEventSdpGetStringLen

Purpose

A string or URL length request has completed. The application initiated this request by calling `BtLibSdpServiceRecordGetStringOrUrlLength()`.

If the `status` field is `btLibErrNoError`, the operation completed successfully, and length can be found in the `eventData` field. Otherwise the operation failed, and the `status` field indicates the reason for the failure.

For this event, the `eventData` field contains the following structure:

Prototype

```
struct {
  BtLibSdpAttributeIdType attributeID;
  BtLibSdpRecordHandle recordH;
  union {
    ...
  } info;
} sdpAttribute;
```

Fields

`attributeID`
: The attribute identifier of the attribute.

`recordH`
: A handle identifying the SDP memory record from which the string length is retrieved.

`info`
: A union containing information specific to the event. See The info Field.

The info Field

For this event, the `info` field contains the following field:

```
UInt16 strLength;
```

This field contains the length of the string or URL represented by the attribute. Bluetooth strings and URLs are not null-terminated.

btLibSocketEventSdpGetNumListEntries

Purpose

A number of list entries request has completed. The application initiated this request by calling `BtLibSdpServiceRecordGetNumListEntries()`.

If the `status` field is `btLibErrNoError`, the operation completed successfully, and the number of list entries can be found in the `eventData` field. Otherwise the operation failed, and the `status` field indicates the reason for the failure.

For this event, the `eventData` field contains the following structure:

Prototype

```
struct {
  BtLibSdpAttributeIdType attributeID;
  BtLibSdpRecordHandle recordH;
  union {
    ...
  } info;
} sdpAttribute;
```

Fields

`attributeID`
: The attribute identifier of the attribute.

`recordH`
: A handle identifying the SDP memory record from which the number of list entries is retrieved.

`info`
: A union containing information specific to the event. See The info Field.

The info Field

For this event, the `info` field contains the following field:

```
UInt16 numItems;
```

This field contains the number of entries in the list attribute.

btLibSocketEventSdpGetNumLists

Purpose

A number of lists request has completed. The application initiated this request by calling `BtLibSdpServiceRecordGetNumLists()`.

If the `status` field is `btLibErrNoError`, the operation completed successfully, and the number of lists can be found in the `eventData` field. Otherwise the operation failed, and the `status` field indicates the reason for the failure.

For this event, the `eventData` field contains the following structure:

Prototype

```
struct {
  BtLibSdpAttributeIdType attributeID;
  BtLibSdpRecordHandle recordH;
  union {
    ...
  } info;
} sdpAttribute;
```

Fields

`attributeID`
: The attribute identifier of the attribute.

`recordH`
: A handle identifying the SDP memory record from which the number of lists is retrieved.

`info`
: A union containing information specific to the event. See The info Field.

The info Field

For this event, the `info` field contains the following field:

```
UInt16 numItems;
```

This field contains the number of lists in the protocol list descriptor attribute.

btLibSocketEventSdpGetRawAttribute

Purpose

A get raw attribute request has completed. The application initiated the request by calling `BtLibSdpServiceRecordGetRawAttribute()`.

If the `status` field is `btLibErrNoError`, the operation completed successfully, and the raw attribute can be found in the `eventData` field. Otherwise the operation failed, and the `status` field indicates the reason for the failure.

For this event, the `eventData` field contains the following structure:

Prototype

```
struct {
  BtLibSdpAttributeIdType attributeID;
  BtLibSdpRecordHandle recordH;
  union {
    ...
  } info;
} sdpAttribute;
```

Fields

`attributeID`
: The attribute identifier of the attribute.

`recordH`
: A handle identifying the SDP memory record from which the raw attribute is retrieved.

`info`
: A union containing information specific to the event. See The info Field.

The info Field

For this event, the `info` field contains the following structure:

Prototype

```
struct {
  UInt16 valSize;
  UInt8 *value;
} rawData;
```

Fields

`valSize`
: Number of size, in bytes, of the raw attribute value.

`value`
: Byte array containing the raw attribute value.

btLibSocketEventSdpGetRawAttributeSize

Purpose

A get raw attribute size request has completed. The application initiated this request by calling `BtLibSdpServiceRecordGetSizeOfRawAttribute()`.

If the `status` field is `btLibErrNoError`, the operation completed successfully, and the size of the attribute can be found in the `eventData` field. Otherwise the operation failed, and the `status` field indicates the reason for the failure.

For this event, the `eventData` field contains the following structure:

Prototype

```
struct {
  BtLibSdpAttributeIdType attributeID;
  BtLibSdpRecordHandle recordH;
  union {
    ...
  } info;
} sdpAttribute;
```

Fields

`attributeID`
: The attribute identifier of the attribute.

`recordH`
: A handle identifying the SDP memory record from which the size of the raw attribute is retrieved.

`info`
: A union containing information specific to the event. See The info Field.

The info Field

For this event, the `info` field contains the following field:

```
UInt16 valSize;
```

This field contains the size, in bytes, of the raw attribute value.

btLibSocketEventSdpGetServerChannelBy Uuid

Purpose A get server channel request has completed. The application initiated this request by calling `BtLibSdpGetServerChannelByUuid()`.

If the status field is `btLibErrNoError`, the operation completed successfully, and the server channel can be found in the `eventData` field. Otherwise, the operation failed, and the `status` field indicates the reason for the failure.

For this event, the `eventData` field contains the following structure:

Prototype

```
struct {
  BtLibSdpRemoteServiceRecordHandle
    remoteHandle;
  union {
    ...
  } param;
} sdpByUuid;
```

Fields

`remoteHandle`
: The handle for the remote SDP service record.

`param`
: A union containing information that depends on the event. See The param Field.

The param Field

For this event, the `param` field contains the following field:

```
BtLibRfCommServerIdType channel;
```

This `BtLibRfCommServerIdType` contains the RFCOMM server channel represented by the SDP service record.

btLibSocketEventSdpGetPsmByUuid

Purpose A get PSM request has completed. The application initiated this request by calling `BtLibSdpGetPSMByUuid()`.

If the status field is `btLibErrNoError`, the operation completed successfully, and the server channel can be found in the `eventData` field. Otherwise, the operation failed, and the `status` field indicates the reason for the failure.

For this event, the `eventData` field contains the following structure:

Prototype

```
struct {
  BtLibSdpRemoteServiceRecordHandle
    remoteHandle;
  union {
    ...
  } param;
} sdpByUuid;
```

Fields

`remoteHandle`
: The handle for the remote SDP service record.

`param`
: A union containing information that depends on the event. See The param Field.

The param Field

For this event, the `param` field contains the following field:

```
BtLibL2CapPsmType psm;
```

This `BtLibL2CapPsmType` contains the PSM value of the L2CAP channel represented by the SDP service record.

btLibSocketEventSendComplete

Purpose

A send request has completed. The application initiated this request by calling `BtLibSocketSend()`.

For this event, the `eventData` field contains the following structure:

Prototype

```
struct {
  UInt16 dataLen;
  UInt8 *data;
} data;
```

Fields

`dataLen`
The number of bytes of data that were actually sent.

`data`
Not used. This variable does not contain any valid information.

Error Conditions

`btLibErrNoError`
Success.

`btLibErrNoAclLink`
No ACL Link.

Socket Disconnection Error Codes

In addition to the standard error codes that can accompany socket events, the status codes accompanying the btLibSocketEventConnectedInbound, btLibSocketEventConnectedOutbound, and btLibSocketEventDisconnected events can have the following additional values:

`btLibL2DiscConfigOptions`
Configuration failed due to an unrecognized configuration option.

`btLibL2DiscConfigReject`
Configuration was rejected (unknown reason).

`btLibL2DiscConfigUnacceptable`
Configuration failed due to unacceptable parameters.

`btLibL2DiscConnNoResources`
The remote device is out of resources.

`btLibL2DiscConnPsmUnsupported`
The remote device does not support the requested protocol service (PSM).

`btLibL2DiscConnSecurityBlock`
The remote device's security system denied the connection.

`btLibL2DiscLinkDisc`
The underlying ACL Link was disconnected.

`btLibL2DiscQosViolation`
: The connection was terminated due to a Quality of Service (QOS) violation.

`btLibL2DiscReasonUnknown`
: Disconnection occurred for an unknown reason.

`btLibL2DiscRequestTimeout`
: An L2CAP request timed out.

`btLibL2DiscSecurityBlock`
: The local security manager refused the connection attempt.

`btLibL2DiscUserRequest`
: Disconnection was requested by either the local or remote device.

Socket Functions

The Bluetooth library uses sockets to represent L2CAP, RFCOMM, and SDP connections. The functions in this section perform general socket tasks and tasks related to L2CAP and RFCOMM sockets. The functions specific to SDP sockets are in the Service Discovery Protocol Functions section.

BtLibSocketAdvanceCredit Function

Purpose Advance credit to a given RFCOMM connection socket.

Declared In `BtLib.h`

Prototype

```
Err BtLibSocketAdvanceCredit (UInt16 btLibRefNum,
   BtLibSocketRef socket, UInt8 credit)
```

Parameters

→ *`btLibRefNum`*
: Reference number for the Bluetooth library.

→ *`socket`*
: RFCOMM socket reference number.

→ *`credit`*
: Number credits to add to the total number of credits for this socket. The total number of credits represents the number of packets the remote device can send before data flow stops.

Returns Returns one of the following values:

`btLibErrNoError`
Success

`btLibErrFailed`
Too many credits advanced.

`btLibErrSocket`
The specified socket is invalid.

`btLibErrSocketProtocol`
The specified socket is not an RFCOMM socket.

`btLibErrSocketRole`
The specified socket is not connected.

Comments RFCOMM uses a credit based flow control mechanism. For each credit the connection has, one packet of data can be sent. When the credits are spent, data flow stops until you advance more credits using this function.

Multiple calls to this function have a cumulative effect.

BtLibSocketClose Function

Purpose Close a socket, free associated resources, and kill all associated socket connections.

Declared In `BtLib.h`

Prototype

```
Err BtLibSocketClose (UInt16 btLibRefNum,
    BtLibSocketRef socket)
```

Parameters

→ *`btLibRefNum`*
Reference number for the Bluetooth library.

→ *`socket`*
Reference number of socket to close.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrSocket`
The specified socket is invalid.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments No callback events are generated when closing a socket.

See Also `BtLibSocketCreate()`, `BtLibSocketListen()`, `BtLibSocketConnect()`, `BtLibSocketRespondToConnection()`

BtLibSocketConnect Function

Purpose Create an outbound L2CAP or RFCOMM connection.

Declared In `BtLib.h`

Prototype

```
Err BtLibSocketConnect (UInt16 btLibRefNum,
    BtLibSocketRef socket,
    BtLibSocketConnectInfoType *connectInfo)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *socket*
Reference number of socket to connect.

→ *connectInfo*
`BtLibSocketConnectInfoType` containing Bluetooth device address and protocol-specific connection information.

Returns Returns one of the following values:

`btLibErrPending`
The results will be returned through a callback event.

`btLibErrNoAclLink`
An ACL link for the remote device does not exist

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrSocket`
The specified socket is invalid.

`btLibErrSocketProtocol`
The protocol of the specified socket is not supported. This function only supports the L2CAP and RFCOMM protocols.

`btLibErrSocketRole`
: The specified socket is already connected or listening.

`btLibErrStackNotOpen`
: The Bluetooth stack failed to initialize when the library was opened.

Comments

If the connection succeeds, the `btLibSocketEventConnectedOutbound` event is generated and its `status` field is set to `btLibErrNoError`. If connection fails, the same event is generated with a non-zero `status` field, or a `btLibSocketEventDisconnected` is generated. In both cases, the `status` field indicates the reason for the failure.

If the connection succeeds, the `btLibSocketEventData` event is generated whenever data is received from the remote device. When the channel disconnects, a `btLibSocketEventDisconnected` event is generated.

BtLibSocketConnectInfoType

The `BtLibSocketConnectInfoType` structure allows you to specify the address of the remote device and data specific to the protocol of the socket. The protocol-specific data is stored as a union; the member of the union that is valid depends on the protocol.

Prototype

```
typedef struct BtLibSocketConnectInfoType {
  BtLibDeviceAddressTypePtr remoteDeviceP;
  union {
    ...
  } data;
} BtLibSocketConnectInfoType;
```

Fields

`remoteDeviceP`
: A pointer to a BtLibDeviceAddressType that contains the address of the remote device.

`data`
: A union containing protocol-specific information. This union has two members: `L2Cap`, and `RfComm`.

L2Cap

Use the `L2Cap` union member if you're setting up a L2CAP socket. This member contains the following structure:

Prototype

```
struct {
  BtLibL2CapPsmType remotePsm;
  UInt16 minRemoteMtu;
  UInt16 localMtu;
} L2Cap;
```

Fields

`remotePsm`
: A `BtLibL2CapPsmType` representing the protocol and service multiplexer (PSM) identifier of the protocol to which this socket should connect. This identifier is obtained using the Service Discovery Protocol (SDP).

`minRemoteMtu`
: The minimum MTU, or packet size, that your application can support.

`localMtu`
: The MTU, or packet size, of the local device.

RfComm

Use the `RfComm` union member if you're setting up a RFCOMM socket. This member contains the following structure:

Prototype

```
struct {
  BtLibRfCommServerIdType remoteService;
  UInt16 maxFrameSize;
  UInt8 advancedCredit;
} RfComm;
```

Fields

`remoteService`
: A `BtLibRfCommServerIdType` representing the RFCOMM service channel on the remote device to which this socket should connect. This identifier is obtained using the Service Discovery Protocol (SDP).

`maxFrameSize`
: The maximum frame size your application can handle. This value must be between `BT_RF_MINFRAMESIZE` and `BT_RF_MAXFRAMESIZE`. If your application can handle any frame size, set this value to `BT_RF_DEFAULT_FRAMESIZE`.

`advancedCredit`
: An amount of credit the socket advances to the remote device when it successfully connects. Additional credit can be

advanced using the `BtLibSocketAdvanceCredit()` function once the connection has been established.

See Also `BtLibSocketSend()`, `BtLibSocketClose()`

BtLibSocketCreate Function

Purpose Create a socket with foreground notification. The Bluetooth library supports a maximum of 16 socket connections.

Declared In `BtLib.h`

Prototype

```
Err BtLibSocketCreate (UInt16 btLibRefNum,
   BtLibSocketRef *socketRefP,
   BtLibSocketProcPtr callbackP, UInt32 refCon,
   BtLibProtocolEnum socketProtocol)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

← *socketRefP*
Pointer to an allocated `BtLibSocketRef` that contains the socket value upon return. This pointer must not be `NULL`.

→ *callbackP*
Callback procedure used to respond to socket events. This value must not be `NULL`.

→ *refCon*
Caller-defined data to pass to the callback procedure.

→ *socketProtocol*
Protocol (L2CAP, RFCOMM, or SDP) to use with this socket.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrParamError`
Either `socketRefP` or `callbackP` is `NULL`.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

`btLibErrTooMany`
The maximum number of sockets allocated for the system has already been reached. The Bluetooth library supports a maximum of 16 socket connections.

Comments No callback events are generated when creating a socket.

Before terminating, applications should close all of the sockets that they have created.

See Also `BtLibSocketConnect()`, `BtLibSocketListen()`, `BtLibSocketClose()`

BtLibSocketGetInfo Function

Purpose Retrieve information for a currently open socket.

Declared In `BtLib.h`

Prototype

```
Err BtLibSocketGetInfo (UInt16 btLibRefNum,
   BtLibSocketRef socket,
   BtLibSocketInfoEnum infoType, void *valueP,
   UInt32 valueSize)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *socket*
Reference number for the socket to query.

→ *infoType*
Type of information to retrieve. See `BtLibSocketInfoEnum`.

← *valueP*
Buffer into which this function stores the result. You must allocate this buffer.

→ *valueSize*
Size, in bytes, of the `valueP` buffer. This size must match the size of the retrieved information.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrNotOpen`
: The referenced Bluetooth library is not open.

`btLibErrParamError`
: One or more parameters is invalid. Be sure that the `valueSize` parameter matches the size of the information you're retrieving.

`btLibErrSdpNotMapped`
: The SDP socket has not been mapped to a remote SDP service record. This error occurs when you try to obtain the SDP service record handle before you map socket to a remote service record using `BtLibSdpServiceRecordMapRemote()`.

`btLibErrSocket`
: The specified socket is invalid or not in use.

`btLibErrSocketRole`
: The specified socket is not connected or has the wrong role for the request.

`btlibErrSocketProtocol`
: The specified socket has the wrong protocol for the request.

`btLibErrStackNotOpen`
: The Bluetooth stack failed to initialize when the library was opened.

Comments **BtLibSocketInfoEnum**

The `BtLibSocketInfoEnum` enum allows you to specify which information you want to retrieve using the `BtLibSocketGetInfo()` function.

Prototype

```
typedef enum {
  btLibSocketInfo_Protocol = 0,
  btLibSocketInfo_RemoteDeviceAddress,
  btLibSocketInfo_SendPending = 100,
  btLibSocketInfo_MaxTxSize,
  btLibSocketInfo_MaxRxSize,
  btLibSocketInfo_L2CapPsm = 200,
  btLibSocketInfo_L2CapChannel,
  btLibSocketInfo_RfCommServerId = 300,
  btLibSocketInfo_RfCommOutstandingCredits,
  btLibSocketInfo_SdpServiceRecordHandle = 400
} BtLibSocketInfoEnum;
```

Constants

`btLibSocketInfo_L2CapChannel`
: `BtLibSocketGetInfo` returns a `BtLibL2CapChannelIDType` that represents the channel identifier for this socket. A `BtLibL2CapChannelIDType` is actually a `UInt16`. This information is valid for L2CAP sockets only. See the "Logical Link Control and Adaptation Protocol Specification" chapter of the *Specification of the Bluetooth System* for more information about channel identifiers.

`btLibSocketInfo_L2CapPsm`
: `BtLibSocketGetInfo` returns a `BtLibL2CapPsmType` that represents the Protocol and Service Multiplexer (PSM) this socket is using to route packets. This information is only valid for L2CAP sockets.

`btLibSocketInfo_MaxRxSize`
: `BtLibSocketGetInfo` returns a `UInt32` representing the maximum packet size the local device can receive.

`btLibSocketInfo_MaxTxSize`
: `BtLibSocketGetInfo` returns a `UInt32` representing the maximum packet size the local device can transmit.

`btLibSocketInfo_Protocol`
: `BtLibSocketGetInfo` returns a `BtLibProtocolEnum` representing the socket's protocol. The members of this enum are `btLibL2CapProtocol`, `btLibRfCommProtocol`, and `btLibSdpProtocol`.

`btLibSocketInfo_RemoteDeviceAddress`
: `BtLibSocketGetInfo` returns a BtLibDeviceAddressType representing the address of the device at the other end of this socket.

`btLibSocketInfo_RfCommServerId`
: `BtLibSocketGetInfo` returns a `BtLibRfCommServerIdType` that represents the socket's RFCOMM server channel. This information is valid for RFCOMM sockets only.

`btLibSocketInfo_RfCommOutstandingCredits`
: `BtLibSocketGetInfo` returns a `UInt16` containing the number of remaining credits on this socket. This information is valid for RFCOMM sockets only.

`btLibSocketInfo_SdpServiceRecordHandle`
: `BtLibSocketGetInfo` returns the `BtLibSdpRemoteServiceRecordHandle` for the service record associated with this socket. This information is valid for SDP sockets only.

`btLibSocketInfo_SendPending`
: `BtLibSocketGetInfo` returns a `Boolean` indicating whether a send is currently in progress.

BtLibSocketListen Function

Purpose Set up an L2CAP or RFCOMM socket as a listener.

Declared In `BtLib.h`

Prototype

```
Err BtLibSocketListen (UInt16 btLibRefNum,
   BtLibSocketRef socket,
   BtLibSocketListenInfoType *listenInfo)
```

Parameters

→ *`btLibRefNum`*
: Reference number for the Bluetooth library.

→ *`socket`*
: Reference number of the socket.

↔ *`listenInfo`*
: Protocol-specific listening information. For more information see `BtLibSocketListenInfoType`. This parameter must not be `NULL`.

Returns Returns one of the following values:

`btLibErrNoError`
: Success. The socket is listening for incoming connections.

`btLibErrBusy`
: The given PSM is in use (L2CAP only)

`btLibErrNotOpen`
: The referenced Bluetooth library is not open.

`btLibErrParamError`
: `listenInfo` is `NULL`.

`btLibErrSocket`
: The specified socket is invalid.

`btLibErrSocketProtocol`
: The protocol of the specified socket is not supported. This function only supports the L2CAP and RFCOMM protocols.

`btLibErrSocketRole`
: The specified socket is already listening or connected.

`btLibErrStackNotOpen`
: The Bluetooth stack failed to initialize when the library was opened.

`btLibErrTooMany`
: There are no resources to create a listener socket of this type.

Comments

A listener socket waits for a remote device to initiate a connection to the local device and then generates a `btLibSocketEventConnectRequest` event to notify the application that it needs to handle the connection attempt.

You need to respond to this event with a call to `BtLibSocketRespondToConnection()` on the listener socket to accept or reject the connection.

Under certain circumstances, the `listenInfo` parameter acts as an output as well as an input. See the documentation for `BtLibSocketListenInfoType` that follows.

BtLibSocketListenInfoType

The `BtLibSocketListenInfoType` structure allows you to specify data specific to the protocol of the listening socket.

Prototype

```
typedef struct BtLibSocketListenInfoType {
  union {
    ...
  } data;
} BtLibSocketListenInfoType;
```

This data is stored in the `data` field, which is a union consisting of two members: `L2Cap`, and `RfComm`. The member of the union that is valid depends on the protocol of the listening socket.

L2Cap

Use the `L2Cap` union member if you're setting up a L2CAP socket as a listener. This member contains the following structure:

Prototype

```
struct {
  BtLibL2CapPsmType localPsm;
  UInt16 localMtu;
  UInt16 minRemoteMtu;
} L2Cap;
```

Fields

`localPsm`
: A `BtLibL2CapPsmType` representing the protocol and service multiplexer (PSM) identifier of the protocol to be used with this socket. You can identify your own protocol provided that its PSM value is odd, is within the range of 0x1001 to 0xFFFF, and has the 9th bit (0x0100) set to zero. These limitations are specified by the *Specification of the Bluetooth System*. If you set this field to `BT_L2CAP_RANDOM_PSM`, the `BtLibSocketListen` function automatically creates a suitable PSM for the channel and returns it in this structure.

`localMtu`
: The maximum transmission unit (MTU), or packet size, of the local device.

`minRemoteMtu`
: The minimum packet size that your application can support.

RfComm

Use the `RfComm` union member if you're setting up a RFCOMM socket as a listener. This member contains the following structure:

Prototype

```
struct {
   BtLibRfCommServerIdType serviceID;
   UInt16 maxFrameSize;
   UInt8 advancedCredit;
} RfComm;
```

Fields

`serviceID`
: A `BtLibRfCommServerIdType` representing the socket's RFCOMM service channel. It is assigned by RFCOMM and returned in this field when you call `BtLibSocketListen`.

`maxFrameSize`
: The maximum frame size your application can handle. This value must be between `BT_RF_MINFRAMESIZE` and `BT_RF_MAXFRAMESIZE`. If your application can handle any frame size, set this value to `BT_RF_DEFAULT_FRAMESIZE`.

`advancedCredit`
An amount of credit the socket advances to the remote device when it successfully connects. Additional credit can be advanced using the `BtLibSocketAdvanceCredit()` function once the connection has been established.

See Also `BtLibSocketClose()`

BtLibSocketRespondToConnection Function

Purpose Accept or reject an in-bound connection on a given listener socket.

Declared In `BtLib.h`

Prototype

```
Err BtLibSocketRespondToConnection
    (UInt16 btLibRefNum, BtLibSocketRef socket,
    Boolean accept)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *socket*
Reference number of the listener socket.

→ *accept*
`true` to accept the connection; `false` to reject the connection.

Returns Returns one of the following values:

`btLibErrNoError`
Success. This status is returned when `accept` is `false`.

`btLibErrFailed`
One or more parameters is invalid.

`btLibErrPending`
The results will be returned through a callback event.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrSocket`
The specified socket is invalid or not in use.

`btLibErrSocketProtocol`
The protocol of the specified socket is not supported. This function only supports the L2CAP and RFCOMM protocols.

`btLibErrSocketRole`
: The specified socket is not a listener socket.

`btLibErrStackNotOpen`
: The Bluetooth stack failed to initialize when the library was opened.

Comments

You should call this function when you respond to a `btLibSocketEventConnectRequest` event delivered to a listener socket.

If the connection succeeds, the `btLibSocketEventConnectedInbound` event is generated and its `status` field is set to `btLibErrNoError`. If connection fails, the same event is generated with a non-zero `status` field, or a `btLibSocketEventDisconnected` is generated. In both cases, the `status` field indicates the reason for the failure.

Once the connection succeeds, a `btLibSocketEventData` callback event is generated whenever data received from the remote device. If the channel disconnects, a `btLibSocketEventDisconnected` is generated.

RFCOMM listener sockets and L2CAP listener sockets behave differently when you call this function. When you respond to an inbound L2CAP connection, a new L2CAP socket is created to exchange data with the remote device, and the L2CAP listener socket continues to listen for more connections. In other words, a single L2CAP listener socket can "spawn" several L2CAP sockets. This mechanism allows you to create a piconet.

On the other hand, when you respond to an RFCOMM connection, the RFCOMM listener socket becomes a connection socket through which you can exchange data with the remote device. If you want to create another RFCOMM connection, you need to create another listener socket.

See Also

`BtLibSocketListen()`, `BtLibSocketSend()`, `BtLibSocketClose()`

BtLibSocketSend Function

Purpose Send data over a connected L2CAP or RFCOMM socket.

Declared In `BtLib.h`

Prototype

```
Err BtLibSocketSend (UInt16 btLibRefNum,
   BtLibSocketRef socket, UInt8 *data,
   UInt32 dataLen)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *socket*
Reference number of the transmitting socket.

→ *data*
Pointer to data to send. If the send returns `btLibErrPending`, the data buffer contents must remain intact until the `btLibSocketEventSendComplete` event occurs.

→ *dataLen*
Length of data to send. This value must be less than the Maximum Transmission Unit (MTU) for the socket. The MTU indicates the size of the largest packet that the remote device can receive and is determined when the socket is connected.

Returns Returns one of the following values:

`btLibErrPending`
The results will be returned through a callback event.

`btLibErrBusy`
A send is already in process.

`btLibErrNoAclLink`
An ACL link for the remote device does not exist

`btlibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrSocket`
The specified socket is invalid.

`btLibErrSocketProtocol`
The protocol of the specified socket is not supported by this function. You can only send using the L2CAP and RFCOMM protocols.

`btLibErrSocketRole`
: The specified socket is not connected.

`btLibErrStackNotOpen`
: The Bluetooth stack failed to initialize when the library was opened.

Comments When the data has been sent successfully, a `btLibSocketEventSendComplete` callback event is generated and its `status` field is set to `btLibErrNoError`. If the data is not sent successfully, the same callback event is generated with a non-zero `status` field.

Note that there can be only one send in progress at a time per socket. You must wait for the `btLibSocketEventSendComplete` event before sending another packet.

See Also `BtLibSocketClose()`

Service Discovery Protocol Functions

This section describes functions and macros related to the Bluetooth Service Discovery Protocol (SDP).

BtLibSdpCompareUuids Function

Purpose Compare two UUIDs.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpCompareUuids (UInt16 btLibRefNum,
   BtLibSdpUuidType *uuid1,
   BtLibSdpUuidType *uuid2)
```

Parameters

→ *btLibRefNum*
: Reference number for the Bluetooth Library.

→ *uuid1*
: UUID to compare.

→ *uuid2*
: UUID to compare.

Returns Returns one of the following values:

`btLibErrNoError`
: UUIDs are the same

`btLibErrError`
: UUIDs are different.

`btLibErrNotOpen`
: The referenced Bluetooth library is not open.

`btLibErrParamError`
: One or both UUIDs are invalid.

`btLibErrStackNotOpen`
: The Bluetooth stack failed to initialize when the library was opened.

BtLibSdpGetPSMByUuid Function

Purpose Get an available L2CAP PSM using SDP.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpGetPsmByUuid (UInt16 btLibRefNum,
   BtLibSocketRef socket,
   BtLibDeviceAddressType *remoteDeviceP,
   BtLibSdpUuidType *serviceUuidList,
   UInt8 uuidListLen)
```

Parameters

→ *`btLibRefNum`*
: Reference number for the Bluetooth library.

→ *`socket`*
: Reference number for an SDP socket.

→ *`remoteDeviceP`*
: Device address of a remote device to query. This parameter must not be `NULL`.

→ *`serviceUuidList`*
: Array of UUIDs that must match those of the service record. This parameter must not be `NULL`.

→ *`uuidListLen`*
: Length of `serviceUuidList`. A maximum of 12 entries is allowed.

Returns Returns one of the following values:

`btLibErrPending`
: The PSM value will be returned through a callback event.

`btLibErrNotOpen`
: The referenced Bluetooth library is not open.

`btLibErrOutOfMemory`
: Not enough memory to complete request

`btLibErrParamError`
: One or more parameters is invalid.

`btLibErrSocket`
: The specified socket is invalid or not in use.

`btLibErrSocketRole`
: The specified socket is not connected.

`btLibErrStackNotOpen`
: The Bluetooth stack failed to initialize when the library was opened.

Comments This function returns the L2CAP PSM of the first SDP record on the remote device that contains all the specified UUIDs.

This function generates a `btLibSocketEventSdpGetPsmByUuid` event when the query completes or fails.

See Also `BtLibSdpGetServerChannelByUuid()`

BtLibSdpGetRawDataElementSize Macro

Purpose Returns a constant representing the data element's size.

Declared In `BtLib.h`

Prototype

```
#define BtLibSdpGetRawDataElementSize (header)
```

Parameters → *`header`*
: First byte of a data element

Returns A constant representing the size of the data element.

Comments The first byte of a SDP data element contains the type and size of the data element.

The result of this macro is one of the following constants:

Data Element Sizes

`btLibDESD_1BYTE`
: A 1-byte element. However, if the data element's type is `btLibDETD_NIL` then the size is 0 bytes.

`btLibDESD_2BYTES`
: A 2-byte element.

`btLibDESD_4BYTES`
: A 4-byte element.

`btLibDESD_8BYTES`
: An 8-byte element.

`btLibDESD_16BYTES`
: A 16-byte element.

`btLibDESD_ADD_8BITS`
: The element's actual data size, in bytes, is contained in the next 8 bits.

`btLibDESD_ADD_16BITS`
: The element's actual data size, in bytes, is contained in the next 16 bits.

`btLibDESD_ADD_32BITS`
: The element's actual data size, in bytes, is contained in the next 32 bits.

These size constants are discussed in greater detail in the "Service Discovery Protocol" chapter of the *Specification of the Bluetooth System*.

See Also BtLibSdpGetRawDataElementType(), BtLibSdpParseRawDataElement(), BtLibSdpVerifyRawDataElement()

BtLibSdpGetRawDataElementType Macro

Purpose Returns an SDP data element's type.

Declared In `BtLib.h`

Prototype

```
#define BtLibSdpGetRawDataElementType (header)
```

→ *`header`*
: The first byte of a data element

Returns The type of the data element.

Comments The first byte of a SDP data element contains the type and size of the data element.

The result of this macro is one of the following constants:

Data Element Types

`btLibDETD_NIL`
Nil, the null type

`btLibDETD_UINT`
Unsigned Integer.

`btLibDETD_SINT`
Signed Integer

`btLibDETD_UUID`
UUID, a universally unique identifier

`btLibDETD_TEXT`
Text string

`btLibDETD_BOOL`
Boolean

`btLibDETD_SEQ`
Data element sequence

`btLibDETD_ALT`
Data element alternative

`btLibDETD_URL`
URL, a uniform resource locator

These types are discussed in greater detail in the "Service Discovery Protocol" chapter of the *Specification of the Bluetooth System*.

See Also `BtLibSdpGetRawDataElementSize()`, `BtLibSdpParseRawDataElement()`, `BtLibSdpVerifyRawDataElement()`

BtLibSdpGetServerChannelByUuid Function

Purpose Get an available RFCOMM server channel using SDP.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpGetServerChannelByUuid
    (UInt16 btLibRefNum, BtLibSocketRef socket,
    BtLibDeviceAddressType *remoteDeviceP,
    BtLibSdpUuidType *serviceUuidList,
    UInt8 uuidListLen)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *socket*
Reference number for an SDP socket.

→ *remoteDeviceP*
Device address of a remote device to query. This parameter must not be `NULL`.

→ *serviceUuidList*
Array of UUIDs that must match those of the service record. This parameter must not be `NULL`.

→ *uuidListLen*
Length of `serviceUuidList`. A maximum of 12 entries is allowed.

Returns Returns one of the following values:

`btLibErrPending`
The server channel will be returned through a callback event.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrOutOfMemory`
Not enough memory to complete request

`btLibErrParamError`
One or more parameters is invalid.

`btLibErrSocket`
The specified socket is invalid or not in use.

`btLibErrSocketRole`
The specified socket is not connected.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments This function returns the RFCOMM server channel number of the first SDP record on the remote device that contains all the specified UUIDs.

This function generates a `btLibSocketEventSdpGetServerChannelBy_Uuid` event when the query completes or fails.

See Also BtLibSdpGetPSMByUuid()

BtLibSdpParseRawDataElement Function

Purpose Parse a raw SDP data element to determine where the data field begins and the size of the data field.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpParseRawDataElement
    (UInt16 btLibRefNum,
    const UInt8 *dataElementP, UInt16 *offset,
    UInt32 *length)
```

Parameters

→ *`btLibRefNum`*
Reference number for the Bluetooth library.

→ *`dataElementP`*
Pointer to a raw SDP data element.

← *`offset`*
Offset, in bytes, between `dataElementP` and the start of the data field.

← *`lengthP`*
Length, in bytes, of the data field.

Returns Returns one of the following values:

`btLibErrNoError`
Successfully parsed the attribute.

`btLibErrNotOpen`
The reference Bluetooth library is not open.

`btLibErrParamError`
: `dataElementP`, `offset`, or `length` is `NULL`.

`btLibErrStackNotOpen`
: The Bluetooth stack failed to initialize when the library was opened.

Comments

A data element has three fields. The first field, called the **header field**, identifies the type of value stored in the data element and the size of the element. The second field, called the **size field**, contains more information about the size of the data if it's not completely specified by the header. Otherwise the size field is omitted. The third field, called the **data field**, contains the data element's actual value.

The offset this function returns is the offset between the start of the data element and the data field. The size this function returns is the the size of the data field. Note that the sum of the offset and the size is the size of the data element.

This function is especially useful for iterating through entries in a list attribute.

The *Specification of the Bluetooth System* has more information about the structure of a data element.

See Also

`BtLibSdpVerifyRawDataElement()`, `BtLibSdpGetRawDataElementType()`, `BtLibSdpGetRawDataElementSize()`

BtLibSdpServiceRecordCreate Function

Purpose

Allocate a memory chunk that represents an SDP service record.

Declared In

`BtLib.h`

Prototype

```
Err BtLibSdpServiceRecordCreate
    (UInt16 btLibRefNum,
    BtLibSdpRecordHandle *recordH)
```

Parameters

→ *`btLibRefNum`*
: Reference number for the Bluetooth library.

← *`recordH`*
: SDP memory handle for the new SDP memory record.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrOutOfMemory`
Not enough memory to allocate the memory chunk.

`btLibErrParamError`
`recordH` is `NULL`.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

See Also BtLibSdpServiceRecordDestroy(), BtLibSdpServiceRecordStartAdvertising(), BtLibSdpServiceRecordStopAdvertising()

BtLibSdpServiceRecordDestroy Function

Purpose Free the memory associated with a SDP memory record.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpServiceRecordDestroy
    (UInt16 btLibRefNum,
    BtLibSdpRecordHandle recordH)
```

Parameters

→ *`btLibRefNum`*
Reference number for the Bluetooth library.

→ *`recordH`*
SDP memory handle associated with the memory chunk to be freed.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrParamError`
: `recordH` does not refer to an valid SDP memory record.

`btLibErrStackNotOpen`
: The Bluetooth stack failed to initialize when the library was opened.

Comments This function stops advertising the record before it frees it.

See Also `BtLibSdpServiceRecordCreate()`, `BtLibSdpServiceRecordStartAdvertising()`, `BtLibSdpServiceRecordStopAdvertising()`

BtLibSdpServiceRecordGetAttribute Function

Purpose Retrieve the value of a specific attribute in a SDP memory record. If the attribute is a list or a protocol descriptor list (a list of lists), this function retrieves the value of a specific list entry.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpServiceRecordGetAttribute
    (UInt16 btLibRefNum,
    BtLibSdpRecordHandle recordH,
    BtLibSdpAttributeIdType attributeID,
    BtLibSdpAttributeDataType *attributeValues,
    UInt16 listNumber, UInt16 listEntry)
```

Parameters

→ *btLibRefNum*
: Reference number for the Bluetooth library.

→ *recordH*
: Handle identifying the SDP memory record.

→ *attributeID*
: Attribute identifier of the attribute to retrieve.

← *attributeValues*
: Buffer into which this function stores the attribute's value. You must allocate this buffer. This pointer must not be `NULL`.

→ *listNumber*
: List to query if the attribute is a protocol descriptor list. Otherwise this parameter is ignored.

→ `listEntry`
Item to get in the list if the attribute is a list attribute. Otherwise this parameter is ignored.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrPending`
The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.

`btLibErrBusy`
The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.

`btLibErrInProgress`
A query is already pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

`btLibErrNoAclLink`
An ACL link to the remote device does not exist.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrOutOfMemory`
Not enough memory to perform the query.

`btLibErrParamError`
`recordH` is an invalid handle or `attributeValues` is `NULL`.

`btLibErrSdpAttributeNotSet`
The specified attribute does not exist in the specified service record.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments If the specified SDP memory record refers to a service record on a remote device, this function generates a `btLibSocketEventSdpGetAttribute` event when the result is available or the query fails. In this case, the buffer to which

`attributeValues` points must not be freed before the event occurs; making the buffer global ensures that it remains over the duration of the SDP query.

If you are retrieving a string or a URL, you need to allocate additional space. See the documentation for `BtLibSdpAttributeDataType` for more information.

This function supports the universal attributes defined in "Service Discovery Protocol" chapter of the *Specification of the Bluetooth System*.

See Also BtLibSdpServiceRecordSetAttribute(), BtLibSdpServiceRecordMapRemote(), BtLibSdpServiceRecordGetNumListEntries(), BtLibSdpServiceRecordGetNumLists(), BtLibSdpServiceRecordGetStringOrUrlLength()

BtLibSdpServiceRecordGetNumListEntries Function

Purpose Get the number of entries in a list attribute.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpServiceRecordGetNumListEntries
    (UInt16 btLibRefNum,
    BtLibSdpRecordHandle recordH,
    BtLibSdpAttributeIdType attributeID,
    UInt16 listNumber, UInt16 *numEntries)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *recordH*
Handle identifying the SDP memory record.

→ *attributeID*
Attribute identifier of the attribute whose number of list entries is retrieved.

→ *listNumber*
List to query if the attribute is a `ProfileDescriptorListEntry`. Otherwise this parameter is ignored.

← *numEntries*
Number of entries in the list.

Returns Returns one of the following values:

`btLibErrNoError`
Success

`btLibErrPending`
The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.

`btLibErrBusy`
The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.

`btLibErrInProgress`
Another query is pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

`btLibErrNoAclLink`
An ACL link to the remote device does not exist.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrOutOfMemory`
Not enough memory to perform this query.

`btLibErrParamError`
`recordH` is an invalid handle or `numEntries` is `NULL`.

`btLibErrSdpAttributeNotSet`
The specified attribute does not exist in the specified service record.

`btLibErrStackNotOpen`
The Bluetooth stack failed to open when the library was opened.

Comments This function supports the universal attributes defined in "Service Discovery Protocol" chapter of the *Specification of the Bluetooth System*. Specifically, this function gives valid results for ServiceClassIdList, ProtocolDescriptorList, BrowseGroupList, LanguageBaseAttributeIDList, and ProfileDescriptorList attributes.

If the specified SDP memory record refers to a service record on a remote device, this function generates a `btLibSocketEventSdpGetNumListEntries` event when the result is available or the query fails.

See Also BtLibSdpServiceRecordGetNumLists(), BtLibSdpServiceRecordGetAttribute(), BtLibSdpServiceRecordGetStringOrUrlLength(), BtLibSdpServiceRecordMapRemote()

BtLibSdpServiceRecordGetNumLists Function

Purpose Get the number of lists in a protocol descriptor list SDP attribute.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpServiceRecordGetNumLists
    (UInt16 btLibRefNum,
    BtLibSdpRecordHandle recordH,
    BtLibSdpAttributeIdType attributeID,
    UInt16 *numLists)
```

Parameters

→ *`btLibRefNum`*
Reference number for the Bluetooth library.

→ *`recordH`*
Handle identifying the SDP memory record.

→ *`attributeID`*
Attribute identifier of the attribute whose number of lists is retrieved.

← *`numLists`*
Number of lists.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrPending`
The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.

`btLibErrBusy`
: The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.

`btLibErrInProgress`
: Another query is pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

`btLibErrNoAclLink`
: An ACL link to the remote device does not exist.

`btLibErrNotOpen`
: The referenced Bluetooth library is not open.

`btLibErrOutOfMemory`
: Not enough memory to perform this query.

`btLibErrParamError`
: `recordH` is an invalid handle or `numLists` is `NULL`.

`btLibErrSdpAttributeNotSet`
: The specified attribute does not exist in the specified service record.

`btLibErrStackNotOpen`
: The Bluetooth stack failed to open when the library was opened.

Comments

If the specified SDP memory record refers to a service record on a remote device, this function generates a `btLibSocketEventSdpGetNumLists` event when the result is available or the query fails.

See Also

`BtLibSdpServiceRecordGetNumListEntries()`, `BtLibSdpServiceRecordGetAttribute()`, `BtLibSdpServiceRecordGetStringOrUrlLength()`, `BtLibSdpServiceRecordMapRemote()`

BtLibSdpServiceRecordGetRawAttribute Function

Purpose Retrieve the value of an attribute of an SDP memory record. The retrieved attribute is in the format defined in the "Service Discovery Protocol" chapter of the *Specification of the Bluetooth System*.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpServiceRecordGetRawAttribute
    (UInt16 btLibRefNum,
    BtLibSdpRecordHandle recordH,
    BtLibSdpAttributeIdType attributeID,
    UInt8 *value, UInt16 *valSize)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *recordH*
Handle identifying the SDP memory record.

→ *attributeID*
Attribute identifier of the attribute to retrieve.

← *value*
Buffer into which this function stores the retrieved SDP attribute data. You must allocate this buffer. This pointer must not be `NULL`.

↔ *valSize*
Size of the `value` buffer upon entry. This parameter must not be zero. Upon return, contains the number of bytes retrieved.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrPending`
The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.

`btLibErrBusy`
The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.

`btLibErrInProgress`
: A query is already pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

`btLibErrNoAclLink`
: An ACL link to the remote device does not exist.

`btLibErrNotOpen`
: The referenced Bluetooth library is not open.

`btLibErrOutOfMemory`
: Not enough memory to perform the query.

`btLibErrParamError`
: `recordH` is an invalid handle, `value` is `NULL`, `valSize` is `0`, or the size of the attribute value is larger than `valSize`.

`btLibErrSdpAttributeNotSet`
: The specified attribute does not exist in the specified service record.

`btLibErrStackNotOpen`
: The Bluetooth stack failed to initialize when the library was opened.

Comments

If the specified SDP memory record refers to a service record on a remote device, this function generates a `btLibSocketEventSdpGetRawAttribute` event when the result is available or the query fails.

See Also

BtLibSdpServiceRecordSetRawAttribute(), BtLibSdpServiceRecordGetSizeOfRawAttribute(), BtLibSdpServiceRecordMapRemote()

BtLibSdpServiceRecordGetSizeOfRawAttribute Function

Purpose Return the size, in bytes, of any attribute of an SDP memory record.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpServiceRecordGetSizeOfRawAttribute
    (UInt16 btLibRefNum,
    BtLibSdpRecordHandle recordH,
    BtLibSdpAttributeIdType attributeID,
    UInt16 *size)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *recordH*
Handle identifying the SDP memory record.

→ *attributeID*
Attribute identifier of the attribute whose size is retrieved.

← *size*
Pointer to a `UInt16` to store the size of the attribute. This parameter must not be `NULL`.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrPending`
The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.

`btLibErrBusy`
The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.

`btLibErrInProgress`
A query is already pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

`btLibErrNoAclLink`
An ACL link to the remote device does not exist.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrOutOfMemory`
Not enough memory to perform the query.

`btLibErrParamError`
`recordH` is an invalid handle or `size` is `NULL`.

`btLibErrSdpAttributeNotSet`
The specified attribute does not exist in the specified service record.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments If the specified SDP memory record refers to a service record on a remote device, this function generates a `btLibSocketEventSdpGetRawAttributeSize` event when the result is available or the query fails.

See Also `BtLibSdpServiceRecordGetRawAttribute()`, `BtLibSdpServiceRecordMapRemote()`, `BtLibSdpServiceRecordSetRawAttribute()`

BtLibSdpServiceRecordGetStringOrUrlLength Function

Purpose Get the length of a string or URL attribute in a SDP memory record.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpServiceRecordGetStringOrUrlLength
    (UInt16 btLibRefNum,
    BtLibSdpRecordHandle recordH,
    BtLibSdpAttributeIdType attributeID,
    UInt16 *length)
```

Parameters

→ *`btLibRefNum`*
Reference number for the Bluetooth library.

→ *`recordH`*
Handle identifying the SDP memory record.

→ *`attributeID`*
Attribute identifier of the attribute whose length is retrieved.

← *length*
Pointer to a `UInt16` where the length of the attribute is stored. This parameter cannot be `NULL`.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrPending`
The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.

`btLibErrBusy`
The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.

`btLibErrInProgress`
A query is already pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

`btLibErrNoAclLink`
An ACL link to the remote device does not exist.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrOutOfMemory`
Not enough memory to perform the query.

`btLibErrParamError`
The `recordH` does not refer to a valid handle, `length` is `NULL`, or the attribute is not a string or a URL.

`btLibErrSdpAttributeNotSet`
The specified attribute does not exist in the specified SDP record.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments Bluetooth strings do not include a null terminator.

If the SDP memory record refers to a service record on a remote device, this function generates a

btLibSocketEventSdpGetStringLen event when the result is available or the query fails.

See Also BtLibSdpServiceRecordGetAttribute(), BtLibSdpServiceRecordGetNumListEntries(), BtLibSdpServiceRecordGetNumLists(), BtLibSdpServiceRecordMapRemote()

BtLibSdpServiceRecordMapRemote Function

Purpose Configure an SDP memory record so it refers to a service record on a remote device.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpServiceRecordMapRemote
   (UInt16 btLibRefNum, BtLibSocketRef socket,
   BtLibDeviceAddressType *remoteDeviceP,
   BtLibSdpRemoteServiceRecordHandle remoteHandle
   , BtLibSdpRecordHandle recordH)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *socket*
Reference number of an SDP socket.

→ *remoteDeviceP*
Device to query.

→ *remoteHandle*
Remote service record handle.

→ *recordH*
SDP memory handle of an empty SDP record.

Returns Returns one of the following values:

`btLibErrNoError`
The mapping was successful.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrOutOfMemory`
Not enough memory to perform mapping.

`btLibErrParamError`
: `recordH` is invalid or refers to an invalid memory chunk.

`btLibErrSdpMapped`
: The SDP memory record is already mapped to a remote service record.

`btLibErrSocket`
: The specified socket is invalid or not in use.

`btLibErrSocketProtocol`
: The specified socket is not an SDP socket.

`btLibErrStackNotOpen`
: The Bluetooth stack failed to initialize when the library was opened.

Comments

You must create an SDP memory record using `BtLibSdpServiceRecordCreate()` before using this function.

Note that this function does not copy the contents of the remote service record to the SDP memory record in local memory.

BtLibSdpServiceRecordSetAttribute Function

Purpose

Set the value of an attribute in an SDP memory record. If the attribute is a list or a protocol descriptor list (a list of lists), this function sets the value of a specific list entry. The SDP memory record must represent a local unadvertised service record.

Declared In

`BtLib.h`

Prototype

```
Err BtLibSdpServiceRecordSetAttribute
    (UInt16 btLibRefNum,
    BtLibSdpRecordHandle recordH,
    BtLibSdpAttributeIdType attributeID,
    BtLibSdpAttributeDataType *attributeValue,
    UInt16 listNumber, UInt16 listEntry)
```

Parameters

→ *btLibRefNum*
: Reference number for the Bluetooth library.

→ *recordH*
: Handle of the service record to modify.

→ *attributeID*
: Attribute identifier of the attribute to set.

→ *attributeValue*
Pointer to the new value for the attribute. This pointer must not be NULL.

→ *listNumber*
List to modify if the attribute is a protocol descriptor list. Otherwise this parameter is ignored.

→ *listEntry*
Item to set in the list if the attribute is a list attribute. Otherwise this parameter is ignored.

Returns Returns one of the following values:

btLibErrNoError
Success.

btLibErrAdvertised
An advertised record was passed in recordH. The record must not be advertised.

btLibErrNotOpen
The referenced Bluetooth library is not open.

btLibErrOutOfMemory
Not enough memory to set the attribute.

btLibErrParamError
recordH is invalid or attributeValue is NULL.

btLibErrRemoteRecord
A remote record was passed in recordH. The record must be local.

btLibErrStackNotOpen
The Bluetooth stack failed to initialize when the library was opened.

Comments This function only works on SDP memory records that are local and not advertised. You can advertise the record after you finish modifying it.

This function supports the universal attributes defined in the *Specification of the Bluetooth System*.

See Also BtLibSdpServiceRecordGetAttribute(), BtLibSdpServiceRecordStartAdvertising(), BtLibSdpServiceRecordStopAdvertising()

BtLibSdpServiceRecordSetAttributesForSocket Function

Purpose Initialize an SDP memory record so it can represent an existing L2CAP or RFCOMM listener socket as a service.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpServiceRecordSetAttributesForSocket
    (UInt16 btLibRefNum, BtLibSocketRef socket,
    BtLibSdpUuidType *serviceUUIDList,
    UInt8 uuidListLen, const Char *serviceName,
    UInt16 serviceNameLen,
    BtLibSdpRecordHandle recordH)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *socket*
Reference number for an RFCOMM or L2CAP socket in listening mode.

→ *serviceUUIDList*
List of UUIDs for the service record.

→ *uuidListLen*
Number of entries in `serviceUUIDList`. A maximum of 12 entries is allowed.

→ *serviceName*
User-friendly name for the service in English.

→ *serviceNameLen*
Size, in bytes, of `serviceName`.

→ *recordH*
Handle of the service record to be initialized.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrAdvertised`
The record specified by `recordH` is being advertised. You must stop advertising the record before you can change it.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrOutOfMemory`
: Not enough memory to store the contents of the SDP record.

`btLibErrParamError`
: `recordH` is not a valid record handle.

`btLibErrRemoteRecord`
: A remote record was passed in `recordH`. Because the service is local, the record must be local.

`btLibErrSocket`
: The specified socket is invalid or not in use.

`btLibErrSocketRole`
: The specified socket is not a listener socket.

`btLibErrStackNotOpen`
: The Bluetooth stack failed to initialize when the library was opened.

Comments You must first create an SDP record using `BtLibSdpServiceRecordCreate()`. However, the record must not be advertised. In other words, don't call `BtLibSdpServiceRecordStartAdvertising()` until after calling this function.

See Also `BtLibSdpServiceRecordCreate()`, `BtLibSocketListen()`

BtLibSdpServiceRecordSetRawAttribute Function

Purpose Set the value for an attribute of a SDP memory record. This function allows you to specify the attribute as an array of bytes in the format defined in the "Service Discovery Protocol" chapter of the

Specification of the Bluetooth System. The SDP memory record must represent a local unadvertised service record.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpServiceRecordSetRawAttribute
    (UInt16 btLibRefNum,
    BtLibSdpRecordHandle recordH,
    BtLibSdpAttributeIdType attributeID,
    const UInt8 *value, UInt16 valSize)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *recordH*
Handle identifying the SDP memory record.

→ *attributeID*
Attribute identifier of the attribute to set.

→ *value*
Array of bytes containing SDP attribute data in the format defined in the SDP protocol. This parameter must not be `NULL`.

→ *valSize*
Size, in bytes, of `value`. This parameter must not be `0`.

Returns Returns one of the following values:

`btLibErrNoError`
Success.

`btLibErrAdvertised`
`recordH` is being advertised. The record must not be advertised.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrOutOfMemory`
Not enough memory to set the attribute.

`btLibErrParamError`
`recordH` is invalid, `value` is `NULL`, or `valSize` is `0`.

`btLibErrRemoteRecord`
`recordH` refers to a service record on a remote device. The service record must be local.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments If the service record is being advertised, you must stop advertising it before you modify it.

See Also `BtLibSdpServiceRecordGetRawAttribute()`, `BtLibSdpServiceRecordSetAttribute()`, `BtLibSdpServiceRecordStopAdvertising()`, `BtLibSdpServiceRecordStartAdvertising()`

BtLibSdpServiceRecordsGetByServiceClass Function

Purpose Get the service record handles corresponding to the service classes advertised on a remote device.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpServiceRecordsGetByServiceClass
    (UInt16 btLibRefNum, BtLibSocketRef socket,
    BtLibDeviceAddressType *remoteDeviceP,
    BtLibSdpUuidType *uuidList,
    UInt16 uuidListLen,
    BtLibSdpRemoteServiceRecordHandle *srvRecList,
    UInt16 *numSrvRec)
```

Parameters

→ *btLibRefNum*
Reference number for the Bluetooth library.

→ *socket*
Reference number of an SDP socket.

→ *remoteDevice*
Remote device to query.

→ *uuidList*
Array of UUIDs identifying the service classes. This parameter must not be `NULL`.

→ *uuidListLen*
Number of elements in the `uuidList`. You can specify a maximum of 12 UUIDs.

← *srvRecList*
: Array of service record handles into which this function stores the results of the SDP query. You must allocate this array. This pointer must not be `NULL`.

↔ *numSrvRec*
: Number of service records allocated in `srvRecList`. This value is sent to the SDP server so it can limit the number of responses. On return, the actual number of records retrieved.

Returns Returns one of the following values:

`btLibErrPending`
: The results will be returned through a callback event.

`btLibErrBusy`
: The connection to the remote device is parked.

`btLibErrInProgress`
: A SDP query is already in progress on this socket.

`btLibErrNoAclLink`
: An ACL link to the remote device does not exist.

`btLibErrNotOpen`
: The referenced Bluetooth library is not open.

`btLibErrOutOfMemory`
: Not enough memory to perform the query.

`btLibErrParamError`
: One or more parameters are invalid.

`btLibErrSocket`
: The specified socket is invalid or not in use.

`btLibErrSocketProtocol`
: The specified socket is not an SDP socket.

`btLibErrStackNotOpen`
: The Bluetooth stack failed to initialize when the library was opened.

Comments You need to allocate `srvRecList`, an array of `BtLibSdpRemoteServiceRecordHandle`s large enough to accommodate all of the service record handles corresponding to the specified service classes. Specify the size of the array using the `numSrvRec` parameter.

This function generates a `btLibSocketEventSdpServiceRecordHandle` event when the matching service records are available or the query fails.

BtLibSdpServiceRecordStartAdvertising Function

Purpose Make visible an SDP memory record representing a local SDP service record. Remote devices can access visible service records through SDP.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpServiceRecordStartAdvertising
    (UInt16 btLibRefNum,
    BtLibSdpRecordHandle recordH)
```

Parameters

→ *`btLibRefNum`*
Reference number for the Bluetooth library.

→ *`recordH`*
Handle of the service record to make available to remote devices.

Returns Returns one of the following values:

`btLibErrNoError`
Success

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrParamError`
`recordH` is not a valid record handle.

`btLibErrRemoteRecord`
`recordH` refers to a remote record. The record must be local.

`btLibErrSdpAdvertised`
The service record is already accessible by remote devices.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments You cannot modify an SDP memory record while it is available to remote devices.

See Also BtLibSdpServiceRecordStopAdvertising()

BtLibSdpServiceRecordStopAdvertising Function

Purpose Hide an SDP memory record representing a local SDP service record. Remote devices cannot access hidden service records through SDP.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpServiceRecordStopAdvertising
    (UInt16 btLibRefNum,
    BtLibSdpRecordHandle recordH)
```

Parameters

→ *`btLibRefNum`*
Reference number for the Bluetooth library.

→ *`recordH`*
Handle of the service record to hide.

Returns Returns one of the following values:

`btLibErrNoError`
Success. The SDP record is no longer available to remote devices.

`btLibErrNotOpen`
The referenced Bluetooth library is not open.

`btLibErrParamError`
`recordH` is not a valid record handle.

`btLibErrRemoteRecord`
`recordH` refers to a remote record. The record must be local.

`btLibErrSdpNotAdvertised`
The service record is already hidden from remote devices.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

See Also `BtLibSdpServiceRecordStartAdvertising()`

BtLibSdpUuidInitialize Macro

Purpose Sets the value of a UUID.

Declared In `BtLib.h`

Prototype

```
#define BtLibSdpUuidInitialize (uuid, value,
    valSize)
```

Parameters

`uuid`
: `BtLibSdpUuidType` to initialize.

`value`
: Array of bytes representing the UUID. The size of this array depends on `valSize`.

`valSize`
: `BtLibSdpUuidSizeEnum` member specifying the size of the `value` array.

Returns None.

BtLibSdpVerifyRawDataElement Function

Purpose Verify that a raw SDP data element is properly formed.

Declared In `BtLib.h`

Prototype

```
Err BtLibSdpVerifyRawDataElement
    (UInt16 btLibRefNum, const UInt8 *value,
    UInt16 valSize, UInt8 maxLevel)
```

Parameters

→ *`btLibRefNum`*
: Reference number for the Bluetooth library.

→ *`value`*
: Raw SDP attribute data.

→ *`valSize`*
: Size of `value`, in bytes. The size of the data element must be less than or equal to this parameter, otherwise this function fails.

→ *`maxLevel`*
: Maximum level of recursion over which this function verifies the data element. Must be at least one.

Returns Returns one of the following values:

`btLibErrNoError`
SDP data element is properly formatted.

`btLibErrError`
SDP data element is not properly formatted.

`btLibErrNotOpen`
The reference Bluetooth library is not open.

`btLibErrParamError`
`value` is `NULL`.

`btLibErrStackNotOpen`
The Bluetooth stack failed to initialize when the library was opened.

Comments

This function checks all size descriptors in the element to ensure that the data element fits into the indicated length. In the case of data element sequences or alternates, this function calls itself recursively.

The `maxLevel` parameter specifies the maximum number of times this function calls itself. Limiting the recursion level prevents an infinite loop if the data is bad. `maxLevel` must be large enough to handle the complete data element. For example, to verify a simple data element such as an unsigned integer, `maxLevel` must be at least 1. To verify a data element sequence of UUIDs, `maxlevel` must be at least 2.

See Also

`BtLibSdpParseRawDataElement()`, `BtLibSdpGetRawDataElementType()`, `BtLibSdpGetRawDataElementSize()`

Application-Defined Functions

This section describes the callback functions that handle socket events. These functions are supplied by the developer and can be named anything.

BtLibSocketCallback Function

Purpose Signal the result of a Bluetooth socket event. When the event takes place, this callback function is called.

Declared In `BtLibTypes.h`

Prototype

```
void (*BtLibSocketProcPtr)
    (BtLibSocketEventType *sEvent, UInt32 refCon)
```

Parameters

→ *sEvent*
: `BtLibSocketEventType` structure containing the event parameters.

→ *refCon*
: General purpose integer which you can use to hold application-specific information. When you call `BtLibSocketCreate()` to create the socket, you can specify a value to pass to this parameter.

Returns Returns nothing.

Comments The event and status of the event are in the `sEvent` structure. See Socket Callback Events for more information.

You must specify this callback function when you create a socket. You do this by calling `BtLibSocketCreate()`.

84

Cryptography Provider Manager

The Cryptography Provider Manager (CPM) is a shared library that acts as a framework for cryptographic services. These services are divided into two levels:

- At the application level, the CPM provides an API that any application can use to perform cryptographic operations.
- The operations themselves are not part of the CPM; instead, they're provided by an **algorithm provider** (or **AP**). To export its functionality to the CPM, an algorithm provider implements a set of callback functions. Third party developers can create and distribute their own APs. The Palm OS provides a default algorithm provider that's used in the absence of alternatives.

The CPM provides the glue between an application and an AP. In general, this glue is invisible: You don't need to know anything about the APs that are available to use their cryptographic services. This makes it possible for an application to use common cryptographic operations without having to be overly concerned with the exact implementation of these operations. It also relieves the algorithm providers from the duty of providing their own invocation API.

NOTE: In Palm OS Garnet, only the application level API is offered; the ability to create and install your own algorithm provider will be supported in a later release.

The Default Provider

In Palm OS Garnet, there is only one algorithm provider. Implemented by RSA, it provides three cryptographic services:

- Data encryption and decryption using an RC4 symmetric key stream cipher
- Message digest creation (hashing) using the SHA-1 algorithm.
- Message verification through a combination of the two other operations.

All of the Palm OS Garnet cryptography operations are provided by RSA.

One of the features of the CPM library is that the developer doesn't need to know anything about cryptography. The functions are designed such that the default settings will yield satisfying results.

Fundamental CPM Functions

The fundamental cryptography functions are:

- `CPMLibEncrypt()` performs a data encryption operation.
- `CPMLibDecrypt()` performs a data decryption operation.
- `CPMLibHash()` creates a message digest.
- `CPMLibVerify()` verifies a message that has a signature and certificate.

These functions operate on blocks of data. For example, `CPMLibEncrypt()` takes a block of data (and a key), encrypts it, and hands back the encrypted data all in one go.

The CPM also provides sets of functions that let you perform these operations sequentially, allowing you to (for example) initialize an encryption "stream," iteratively feed data to the encryption algorithm, and then "finalize" the stream and get the encrypted data back. These functions are (for encryption) `CPMLibEncryptInit()`, `CPMLibEncryptUpdate()`, and `CPMLibEncryptFinal()`.

Using the Crypto-Info Structures

The cryptography operations that are currently supported by the CPM library rely on four "crypto-info" data structures:

- The `APKeyInfoStruct` describes a symmetric key,
- `APCipherInfoStruct` describes an encryption/decryption operation.
- `APHashInfoStruct` describes message digest creation.
- `APVerifyInfoStruct` describes a message verification operation.

When you perform a cryptographic operation, you'll be asked for one or more of these structures. In most cases you can pass in an "empty" structure and the function will populate it for you. However, it's important that you zero the structure first.

For example, the `CPMLibHash()` function takes a `APHashInfoStruct` as an argument. To use the default hashing operations, simply allocate the structure, zero it, and pass it in:

```
APHashInfoType hashinfo;
MemSet ((void *)&hashinfo, sizeof (APHashInfoType), 0);
CPMLibHash( ..., &hashInfo, ...);
```

When you're done with a crypto-info structure, you must "release" it:

```
CPMLibReleaseHashInfo( ..., &hashInfo );
```

Using the Export Functions

The cryptography structures (key, hash, cipher, verification) can be "exported," or encoded into a form that can be cached. These operations are provided by the `CPMLibExport...Info` functions. Purposeful details of exporting (and importing, its functional complement) are given in the individual function descriptions, below. This little section touches on a wrinkle of usage.

When you use an export function, you're asked to supply a buffer that can accommodate the encoded data. This means that you have to know how big the encoded data will be; the export function itself tells you the size through its final (reference) argument. Thus, you have to call the export function twice: Once to get the buffer size, and then again to actually get the encoded data. This is demonstrated below:

```
UInt32 length=0;
UInt8 *data=NULL;
Err error;

/* For demo purposes, we're only interested in the last two
 * arguments (the buffer and the buffer length). First, we
 * set the buffer to NULL, and retrieve the length.
 */
error = CPMLibExport... (... NULL, &length);

/* At this pass, we expect the function to tell us that
 * the buffer is too small. Any other return is treated
 * as an error. Note that the <length> argument is reset to
 * to the required allocation length despite the error
 * return.
 */
if (error != cpmErrBufTooSmall)
  return ...; // or whatever

/* Allocate the buffer (for brevity, the example omits
 * the error check).
 */
data = MemPtrNew(length);

/* Call the export function again to retrieve the
 * encoded data.
 */
error = CPMLibExport... ( ... data, &length);

/* This time we want a 'clean' return. */
if (error != errNone) {
  // handle the error
}
```

CPM and AP Constants

AP Capability Constants

The AP capability values are bitfield constants that represent the capabilities that an AP supports. The capability constants are `OR`'d into the `flags` field of the `APProviderInfoStruct` structure.

Declared in `CPMLibCommon.h`, the constants are:

`APF_MP`
: Streaming operations (initialize/update/finalize) are supported ("MP" stands for "multiple part"). See "Fundamental CPM Functions" for details. Note that all APs are expected to support block operations.

`APF_HW`
: The AP's algorithms are implemented in hardware (such as a SmartCard).

`APF_KEYGEN`
: Symmetric key operations (including key info import and export) are supported.

`APF_KEYPAIRGEN`
: Asymmetric key operations (including key info import and export) are supported.

`APF_KEYDERIVE`
: Key derivation is supported.

`APF_HASH`
: Hashing operations (including hash info import and export) are supported.

`APF_CIPHER`
: Message encryption and decryption operations (including cipher info import and export) are supported.

`APF_SIGN`
: Message signing operations (including sign info import and export) are supported.

`APF_VERIFY`
: Message verification operations (including verify info import and export) are supported.

Block Encryption Mode Constants

Purpose The constants listed below represent the various block encryption modes that may be supported by an AP. You can request a particular mode by setting the APCipherInfoStruct `mode` field before passing the structure to the `CPMLibEncrypt()` or `CPMLibDecrypt()`

function. If you don't specify a mode, the AP will choose one for you, and reset the `mode` field to the chosen mode.

The `APModeEnum` data type is used to type the encryption mode constants.

Prototype

```
typedef UInt32 APModeEnum;
```

Note that encryption modes apply to block operations only. Specifying a mode for a stream encryption (through `CPMLibEncryptInit()` *et al.*) has no effect.

The constants (and the type) are declared in CPMLibConstants.h. The constants are:

Constants

`apModeTypeUnspecified = 0`
: The mode isn't specified; the AP uses its default mode.

`apModeTypeNone`
: The encryption mode doesn't apply.

`apModeTypeECB`
: Electronic codebook mode.

`apModeTypeCBC`
: Cipher block chaining mode.

`apModeTypeCBC_CTS`
: Cipher block chaining with cipher text stealing.

`apModeTypeCFB`
: Cipher feedback mode.

`apModeTypeOFB`
: Output feedback mode.

`apModeCounter`
: Counter mode

Cipher Algorithm Constants

Purpose

These constants represent the different cipher algorithms that an AP may support. The algorithm type is encoded in the `type` field of the `APKeyInfoStruct` structure. You can request a specific algorithm by setting the value of the `type` field before passing the structure to `CPMLibGenerateKey()`. If you don't care, leave the field zero'd

(`apAlgorithmTypeUnspecified`); the function will set the field to tell you which algorithm was used.

In the `APKeyInfoStruct` structure, the constants are typed as `APAlgorithmEnum`:

Prototype

```
typedef UInt32 APAlgorithmEnum;
```

The constants (and the type) are defined in `CPMLibCommon.h`. There we see the usual "unspecified" constant:

`apAlgorithmTypeUnspecified = 0`
: No algorithm specified; the AP will use its default.

The rest of the constants are divided into groups, below, and listed with very little additional explanation: The constants' names are reasonably self-documenting.

Block Cipher Algorithms

`apSymmetricTypeDES`
`apSymmetricTypeRC2`
`apSymmetricTypeRC4`
`apSymmetricTypeRC5`
`apSymmetricTypeRC6`,
`apSymmetricTypeDESX_XDX3` ("Strong" DES)
`apSymmetricType3DES_EDE2`
`apSymmetricType3DES_EDE3`
`apSymmetricTypeIDEA`
`apSymmetricTypeDiamond2`
`apSymmetricTypeBlowfish`
`apSymmetricTypeTEA` (Tiny Encryption Algorithm),
`apSymmetricTypeSAFER` (Safe and Fast Encryption Routine),
`apSymmetricType3WAY`
`apSymmetricTypeGOST` (USSR Government Standard),
`apSymmetricTypeSHARK`
`apSymmetricTypeCAST128`, `apSymmetricTypeSquare`,
`apSymmetricTypeSkipjack`

Stream Ciphers

`apSymmetricTypePanama`, `apSymmetricTypeARC4`,
`apSymmetricTypeSEAL`, `apSymmetricTypeWAKE`,
`apSymmetricTypeSapphire`, `apSymmetricTypeBBS`

AES Block Ciphers

`apSymmetricTypeRijndael`, `apSymmetricTypeCAST256`, `apSymmetricTypeTwofish`, `apSymmetricTypeMARS`, `apSymmetricTypeSerpent`

Asymmetric Key Ciphers

`apAsymmetricTypeRSA`, `apAsymmetricTypeDSA`, `apAsymmetricTypeElgamal`, `apAsymmetricTypeNR` (Nyberg-Rueppel), `apAsymmetricTypeBlumGoldwasser`, `apAsymmetricTypeRabin`, `apAsymmetricTypeRW` (Rabin-Williams), `apAsymmetricTypeLUC`, `apAsymmetricTypeLUCELG`, `apAsymmetricTypeECDSA`, `apAsymmetricTypeECNR`, `apAsymmetricTypeECIES`, `apAsymmetricTypeECDHC`, `apAsymmetricTypeECMQVC`

Key Agreement Ciphers

`apKeyAgreementTypeDH` (Diffie-Hellman), `apKeyAgreementTypeDH2` (Unified Diffie-Hellman), `apKeyAgreementTypeMQV` (Menezes-Qu-Vanstone), `apKeyAgreementTypeLUCDIF`, `apKeyAgreementTypeXTRDH`

Export Encoding Constants

Constants that represent different data encoding schemes that are used to convert the crypto-info structs (APKeyInfoStruct, APHashInfoStruct, et al.) into a form that can be cached.

The encoding formats are used by the import and export functions, `CPMLibImportKeyInfo()`, `CPMLibExportKeyInfo()`, `CPMLibImportHashInfo()`, `CPMLibExportHashInfo()`, `CPMLibImportVerifyInfo()`, `CPMLibExportVerifyInfo()`, `CPMLibImportCipherInfo()`, and `CPMLibExportCipherInfo()`.

Declared in `CPMLibCommon.h`, the constants are:

`IMPORT_EXPORT_TYPE_RAW`
: The default encoding, as defined by the AP.

`IMPORT_EXPORT_TYPE_DER`
: ASN.1 DER encoding.

`IMPORT_EXPORT_TYPE_XML`
: Standardized XML encoding.

Hashing Algorithm Constants

The hashing constants represent various hashing algorithms. If you want to tell the AP to use a specific algorithm in a hashing operation, you would pass one of these constants as an argument to `CPMLibHash()` or `CPMLibHashInit()`. If you want the default, use `apHashTypeUnspecified`, The algorithm that's actually used is returned through a `APHashInfoStruct` structure.

Declared in `CPMLibCommon.h`, the constants are:

`apHashTypeUnspecified = 0`
: Unspecified hash algorithm; when generating a message digest, the AP decides which algorithm to use.

`apHashTypeNone`
: Don't hash. Use this constant if you want the AP to suppress hashing in an operation, such as verification, that normally performs it

`apHashTypeMD2`
: Rivest Message Digest 2 (MD2)

`apHashTypeMD5`
: Rivest Message Digest 5 (MD5)

`apHashTypeSHA1`
: Secure Hash Algorithm-160 (SHA-1)

`apHashTypeSHA256`
: SHA 256-bit algorithm

`apHashTypeSHA384`
: SHA 384-bit algorithm

`apHashTypeSHA512`
: SHA 512-bit algorithm

`apHashTypeHAVAL`
: HAVAL one-way algorithm

`apHashTypeRIPEMD160`
: RIPEMD 160-bit

`apHashTypeTiger`
: Tiger algorithm

`apHashTypePanama`
: PANAMA algorithm

Key Class Constants

Purpose

A key's "class" specifies whether the key is symmetric or asymmetric and, if the latter, whether it's public or private. The key class constants represent these qualities. Key class is encoded in the `keyclass` field of the `APKeyInfoStruct` structure.

In the struct, the constants are typed as `APKeyClassEnum`:

Prototype

```
typedef UInt32 APKeyClassEnum;
```

The constants (and the type) are defined in `CPMLibCommon.h`. The constants are:

Constants

`apKeyClassUnspecified = 0`
: The key's class is unspecified.

`apKeyClassSymmetric`
: This is a symmetric key.

`apKeyClassPublic`
: This is the public part of an asymmetric key.

`apKeyClassPrivate`
: This is the private part of an asymmetric key.

Key Usage Constants

Purpose

NOTE: The key usage constants are currently unused.

The key usage constants describe the different ways that an encryption key can be used. How a key is used is encoded in the `usage` field of the key's `APKeyInfoStruct` structure.

Note that the key usage values are mutually exclusive; you can't OR a set of key usage constants to create a "selectively talented" key. The `apKeyUsageAll` constant is the only "multi-purpose" value currently provided.

The `APKeyUsageEnum` data type is used to type the key usage constants:

Prototype

```
typedef UInt32 APKeyUsageEnum;
```

The constants (and the type) are defined in `CPMLibConstants.h`. The constants are:

Constants

`apKeyUsageUnspecified = 0`
: The key's usage is unspecified.

`apKeyUsageAll`
: The key can be used in any operation.

`apKeyUsageCertificateSigning`
: The key is intended for certificate signing.

`apKeyUsageSigning`
: The key is intended for message signing operations.

`apKeyUsageEncryption`
: The key is intended for key or data encryption operations.

`apKeyUsageKeyEncrypting`
: The key is intended for key encryption operations

`apKeyUsageDataEncrypting`
: The key is intended for data encryption operations.

`apKeyUsageMessageIntegrity`
: The key is intended for message verification operations.

Plaintext Padding Constants

Purpose

The plaintext padding constants describe the different ways that plaintext is padded before it's encrypted. The padding type is encoded in the `padding` field of an `APCipherInfoStruct` structure.

The `APPaddingEnum` data type is used to type the padding constants:

Prototype

```
typedef UInt32 APPaddingEnum;
```

The constants (and the type) are declared in `CPMLibCommon.h`. The constants are:

Constants

`apPaddingTypeUnspecified = 0`
: The padding is unspecified; the AP will use the default

`apPaddingTypeNone`
: Specifically request that padding *not* be applied.

`apPaddingTypePKCS1Type1`
: Public Key Cryptography Standard 1, type 1

`apPaddingTypePKCS1Type2`
: Public Key Cryptography Standard 1, type 2

`apPaddingTypePKCS5`
: Public Key Cryptography Standard 5

`apPaddingTypeOAEP`
: Optimal Asymmetric Encryption Padding

`apPaddingTypeSSLv23`
: Secure Sockets Layer version 23

CPM Error Codes

The table below lists and explains the error codes that are returned by the CPM functions.

`cpmErrAlreadyOpen`
: Returned by CPMLibOpen() to indicate that the CPM library has already been opened by your application. The open library remains open.

`cpmErrNotOpen`
: All CPM functions (except CPMLibOpen()) expect the CPM library to be open. This code is returned if the library isn't open. To open the library, call CPMLibOpen().

`cpmErrStillOpen`
: Returned by CPMLibClose() if the function was unable to close the library.

`cpmErrNoProviders`
: Returned by CPMLibOpen() if the function was uable to locate (or load) any crytography providers. Without a

cryptography provider, none of the other CPM functions will work.

`cpmErrNoBaseProvider`
: Returned by CPMLibOpen() if the function was uable to locate (or load) the default cryptography provider. (Note that this means that at least one non-default provider was found. If no providers were found, the function would have returned cpmErrNoProviders).

`cpmErrProviderNotFound`
: Returned by CPMLibGenerateKey(), CPMLibEncryt(), CPMLibDecrypt() and other algorithm-dependent functions if the requested provider couldn't be found.

`cpmErrParamErr`
: Returned by a number of CPM functions when an argument is invalid (a pointer that points to an unallocated structure; a structure field that isn't properly set, and so on).

`cpmErrOutOfMemory`
: Returned by CPMLibOpen() if the memory for a new library handle (and the structures it represents) couldn't be allocated

`cpmErrBufTooSmall`
: Returned by CPMLibExportKey() and CPMLibExportContext() functions if the storage buffer (allocated by the caller) isn't big enough to accommodate the key or context.

`cpmErrBadData`
: Returned by algorithm functions (CPMLibEncrypt...(), CPMLibDecrypt...(), etc.) when the key or other required data is invalid.

`cpmErrUnimplemented`
: Returned by functions that aren't curently implemented.

`cpmErrUnsupported`
: Returned by functions that aren't currently supported.

`cpmErrNoGlobals`
: Returned by functions that access global CPM data—functions such as CPMLibEnumerateProviders() and CPMLibGetProviderInfo()—when that data doesn't exist.

`cpmErrKeyExists`
: Returned by CPMLibImportKey() when the key you're trying to import already exists.

`cpmErrKeyNotFound`
: Returned by CPMLibExportKey() when the key you're trying to export doesn't exists.

CPM and AP Structures and Data Types

APCipherInfoStruct Struct

Purpose

The `APCipherInfoStruct` encapsulates information about an instance of a data encryption or decryption operation. The structure is populated and returned by the data encryption and decryption functions (`CPMLibEncrypt()`, `CPMLibDecrypt()`, `CPMLibEncryptInit()`, and `CPMLibDecryptInit()`). You can set some of the fields' values yourself (before calling an encryption/decryption function) to fine-tune the impending operation.

You're responsible for allocating and freeing the `APCipherInfoStructs` that you need—the CPM never allocates them for you. The `APCipherInfoStructs` that you create (and actually use) must be released through `CPMLibReleaseCipherInfo()` before they're freed.

For more information (including examples) on how to use a crypto-info structure, see "Using the Crypto-Info Structures."

Declared in `CPMLibCommon.h`, the structure looks like this:

Prototype

```
struct APCipherInfoStruct {
   APProviderContextType  providerContext;
   APAlgorithmEnum  type;
   APPaddingEnum  padding;
   UInt8  *iv;
   UInt32  ivLength;
   void  *algorithmParams;
};
```

Fields

`providerContext`
: Information about the AP that performed (or is requested to perform) the operation. See `APProviderContextStruct`.

`type`
: Constant that represents the cipher algorithm that was used or that's requested. See Cipher Algorithm Constants

`padding`
: Constant that represents the plaintext padding scheme used by the algorithm. See Plaintext Padding Constants.

`iv`
: Initialization vector.

`ivLength`
: The length of the initialization vector, in bytes.

`algorithmParams`
: Additional data that's fed to the algorithm, as specified by the AP.

APHashInfoStruct Struct

Purpose

The APHashInfoStruct contains information about an instance of a message digest operation. You allocate the structure yourself, zero its contents, set the fields that you're interested in (if any), and then pass it as an argument to the message digest functions (`CPMLibHash()`, `CPMLibHashInit()`, `CPMLibExportHashInfo()`, and so on). When you're finished with the struct, you pass it to `CPMLibReleaseHashInfo()`.

For more information (including examples) on how to use the structure, see "Using the Crypto-Info Structures."

Declared in CPMLibCommon.h, the structure looks like this:

Prototype

```
struct APHashInfoStruct {
  APProviderContextType  providerContext;
  APHashEnum  type;
  UInt32  length;
};
```

Fields

`providerContext`
: Information about the AP that performed (or is requested to perform) the operation. See `APProviderContextStruct`.

`type`
: The hashing algorithm that was used to create (or is requested to create) the message digest.

`length`
The length of the digest.

Comments The APHashInfoStruct is one of the crypto-info structures; it's used as an input/output argument to the message digest functions (`CPMLibHash()`, `CPMLibHashInit()`, `CPMLibExportHashInfo()`, and so on). You allocate the structure yourself; before passing the struct to a function, you *must* zero its contents. When you're finished with the struct, you pass it to `CPMLibReleaseHashInfo()`. For more information on how to use a crypto-info structure, see "Using the Crypto-Info Structures."

APKeyInfoStruct Struct

Purpose The `APKeyInfoStruct` holds information about an encrytion key.

Prototype

```
struct APKeyInfoStruct {
  APProviderContextType  providerContext;
  APAlgorithmEnum  type;
  APKeyUsageEnum  usage;
  APKeyClassEnum  keyclass;
  UInt32  length;
  UInt32  actualLength;
  UInt16  exportable;
  UInt16  ephemeral;
};
```

Fields

`providerID`
Algorithm provider ID number.

`type`
A code that identifies the type of algorithm, one of the Cipher Algorithm Constants values.

`usage`
A code that identifies how the algorithm is used, one of the Key Usage Constants values.

`keyLength`
Length of the key data, in bytes and padded to the next largest word. (Default is 8.)

`keyActualLength`
The actual, unpadded length of the key data. (Default is 7).

`exportable`
: Can this key be used in a CPMExportKey() call?. 1 if it can, 0 if it can't.

`ephemeral`
: Is this a "one-shot" key? 1 if it is, 0 if it's permanent..

APProviderContextStruct Struct

Purpose

The `APProviderContextStruct` contains information about an instance of a cryptographic operation. The structure is contained (as an `APProviderContextType` value) in the four crypto-info structs, `APKeyInfoStruct`, `APHashInfoStruct`, `APCipherInfoStruct`, and `APVerifyInfoStruct`.

Declared in `CPMLibCommon.h`, the structure looks like this:

Prototype

```
struct APProviderContextStruct {
  UInt32  providerID;
  void  *localContext;
};
```

Fields

`providerID`
: Integer that uniquely identifies the provider that's being used in the operation.

`localContext`
: Provider-specific information about the operation.

APProviderInfoStruct Struct

Purpose

The `APProviderInfoStruct` contains information about a specific algorithm provider. The structure is returned (as an `APProviderInfoType`) by `CPMLibGetProviderInfo()`.

Declared in `CPMLibCommon.h`, the structure looks like this:

Prototype

```
struct APProviderInfoStruct {
  char  name[32];
  char  other[64];
  UInt32  flags;
  UInt8  numAlgorithms;
  Boolean  bHardware;
};
```

Fields

`name`
The human-readable name of the provider.

`other`
Additional textual information.

`flags`
A bitfield that publishes the functionality that this provider supports. See AP Capability Constants for a list of values that this field can combine.

`numAlgorithms`
A count of the algorithms this provider supplies.

`bHardware`
`true` if the provider's algorithms are implemented in hardware; `false` if in software.

APVerifyInfoStruct Struct

Purpose

The `APVerifyInfoStruct` is used by the verification functions (`CPMLibVerify()`, *et al.*) to verify a message. It contains (primarily) the hash operation and cipher operation information that will be used during verification. It's the caller's responsibility to allocate and embed the structure's `APHashInfoStruct` and `APCipherInfoStruct` fields before passing the `APVerifyInfoStruct` to a verification function.

Any `APVerifyInfoStructs` that you actually use must be released through `CPMLibReleaseVerifyInfo()` before it's freed. The embedded structures must also be released through `CPMLibReleaseCipherInfo()` and `CPMLibReleaseHashInfo()`.

For more information (including examples) on how to use a crypto-info structure, see "Using the Crypto-Info Structures."

Declared in `CPMLibCommon.h`, the structure looks like this:

Prototype

```
struct APVerifyInfoStruct {
  APProviderContextType  providerContext;
  APHashInfoType  *hashInfoP;
  APCipherInfoType  *cipherInfoP;
}
```

Fields

`providerContext`
Information about the AP that performed (or is requested to perform) the operation. See `APProviderContextStruct`.

`hashInfoP`
`APHashInfoStruct` that contains the certificate's hash information.

`cipherInfoP`
`APCipherInfoStruct` that contains the certificate's cipher information.

CPMInfoStruct Struct

Purpose

Structure that provides information about the CPM library. It's used by the `CPMLibGetInfo()` function.

Prototype

```
typedef struct CPMInfoStruct {
  UInt8  numInstances;
  UInt8   numProviders;
  Boolean  defaultProviderPresent;
};
```

Fields

`numInstances`
The number of clients (applications) that are talking to this library.

`numProviders`
The number of algorithm providers that this library knows about.

`defaultProviderPresent`
Does the library contain a default provider? `true` if it does, otherwise `false`.

CPM Functions

CPMLibDecrypt Function

Purpose Decrypts a block of encrypted data.

Declared In `CPMLib68kInterface.h, CPMLibARMInterface.h`

Prototype

```
Err CPMLibDecrypt ( UInt16 libRef,
   APKeyInfoType *keyInfo,
   APCipherInfoType *cipherInfo, UInt8 *inBuffer,
   UInt32 inBufferLength, UInt8 *outBuffer,
   UInt32 *outBufferLength )
```

Parameters

→ *libRef*
(68k only) CPM Library reference number.

→ *keyInfo*
`APKeyInfoStruct` that contains the key that was used to encrypt the data. You obtain the structure by importing the `APKeyInfoStruct` that was exported by the data encryptor.

↔ *cipherInfo*
A pointer to an `APCipherInfoStruct` that describes the parameters that will be used in the decryption operation. You typically retrieve the structure by importing the `APCipherInfoStruct` that was exported by the data encryptor. If you're using the decryption defaults provided by the CPM library, you can pass in a freshly allocated (and zero'd) `APCipherInfoStruct`; in this case, the structure will be populated by this function to reflect the actual decryption parameters.

→ *inBuffer*
A pointer to the (encrypted) data that you want to decrypt.

→ *inBufferLength*
Length, in bytes, of `inBuffer`.

← *outBuffer*
A pointer to the buffer into which the decrypted data will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the decrypted data.

↔ *outBufferLength*
You pass in the (allocated) length of `outBuffer`, in bytes. The function resets the argument to the amount of data that was actually copied into `outBuffer`. If the function returns `cpmErrBufTooSmall`, `outBufferLength` is set to the minimum buffer size that's needed to accommodate the decrypted data..

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments This functions peforms block decryption. For stream decryption, see `CPMLibDecryptInit()`.

The `keyInfo` and `cipherInfo` must agree on the algorithm type, as specified in their respective `APAlgorithmEnum` fields.

If `outBuffer` isn't big enough, the function will fail and `outBufferLength` will return the "correct" output buffer size (i.e. large enough to accommodate the encrypted data) . When this happens, simply reallocate the output buffer and call `CPMLibDecrypt` again.

CPMLibDecryptFinal Function

Purpose Finalizes a stream decryption operation.

Declared In `CPMLib68kInterface.h, CPMLibARMInterface.h`

Prototype

```
Err CPMLibDecryptFinal ( UInt16 libRef,
    APKeyInfoType *keyInfo,
    APCipherInfoType *cipherInfo, UInt8 *inBuffer,
    UInt32 inBufferLength, UInt8 *outBuffer,
    UInt32 *outBufferLength)
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *keyInfo*
Key used to decrypt the data, as returned by `CPMLibGenerateKey()` or `CPMLibImportKeyInfo()`.

→ *cipherInfo*
A pointer to the `APCipherInfoStruct` that was returned by the stream-initializing `CPMLibDecryptInit()` call.

→ *inBuffer*
: A pointer to the data that you want to encrypt. If you already supplied all the data through previous `CPMLibDecryptUpdate()` calls, pass `NULL`.

→ *inBufferLength*
: The length of `inBuffer`, in bytes. If `inBuffer` is `NULL`, pass 0.

← *outBuffer*
: A pointer to the buffer where the decrypted `inBuffer` data will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the decrypted data.

↔ *outBufferLength*
: You pass in the (allocated) length of `outBuffer`, in bytes. The function resets the argument to the amount of data that was actually copied into `outBuffer`. If the function returns `cpmErrBufTooSmall`, `outBufferLength` is set to the minimum buffer size that's needed to accommodate the decrypted `inBuffer` data..

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments This function (optionally) decrypts a final buffer of data and then closes the decryption stream that was initialized by `CPMLibDecryptInit()`.

Note that `outBuffer` contains the decrypted `inBuffer` data only—it doesn't contain all the data that was decrypted by this stream. It's the callers responsibility to accumulate the data that was decrypted by previous, successive `CPMLibDecryptUpdate()` calls.

CPMLibDecryptInit Function

Purpose Initializes a stream decryption session.

Declared In `CPMLib68kInterface.h, CPMLibARMInterface.h`

Prototype

```
Err CPMLibDecryptInit ( UInt16 libRef,
   APKeyInfoType *keyInfo,
   APCipherInfoType *cipherInfo )
```

Parameters

→ *libRef*
: CPM Library reference number (68k only).

→ *keyInfo*
: `APKeyInfoStruct` that contains the key that was used to encrypt the data. You obtain the structure by importing the `APKeyInfoStruct` that was exported by the data encryptor.

↔ *cipherInfo*
: A pointer to an `APCipherInfoStruct` that describes the parameters that will be used in the decryption operation. You typically retrieve the structure by importing the `APCipherInfoStruct` that was exported by the data encryptor. If you're using the decryption defaults provided by the CPM library, you can pass in a freshly allocated (and zero'd) `APCipherInfoStruct`; in this case, the structure will be populated by this function to reflect the actual decryption parameters.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments This function initializes a stream decryption operation. To feed data to the operation, you call `CPMLibDecryptUpdate()` followed by `CPMLibDecryptFinal()`. The "update" function is optional; the "final" function is mandatory. For block encryption, see `CPMLibDecrypt()`.

The `keyInfo` and `cipherInfo` must agree on the algorithm type, as specified in their respective `APAlgorithmEnum` fields.

CPMLibDecryptUpdate Function

Purpose Feeds data to a stream decryption operation.

Declared In `CPMLib68kInterface.h, CPMLibARMInterface.h`

Prototype

```
Err CPMLibDecryptUpdate ( UInt16 libRef,
    APKeyInfoType *keyInfo,
    APCipherInfoType *cipherInfo, UInt8 *inBuffer,
    UInt32 inBufferLength, UInt8 *outBuffer,
    UInt32 *outBufferLength)
```

Parameters see `CPMLibEncryptFinal()`

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments This function feeds data into the stream decryption session that was started by `CPMLibDecryptInit()`. You can make any number of `CPMLibDecryptUpdate` calls while the stream is open. When you've finished feeding data into the stream, you call `CPMLibDecryptFinal()`.

This function's arguments, return values, and behavior are nearly identical to `CPMLibDecryptFinal()` (which see for details). The only difference between them is that this function leaves the stream open, and `CPMLibDecryptFinal()` closes it.

CPMLibEncrypt Function

Purpose Encrypts a block of data.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibEncrypt ( UInt16 libRef,
    APKeyInfoType *keyInfo,
    APCipherInfoType *cipherInfo, UInt8 *inBuffer,
    UInt32 inBufferLength, UInt8 *outBuffer,
    UInt32 *outBufferLength)
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *keyInfo*
A pointer to an `APKeyInfoStruct` that represents the key that will be used to encrypt the data. The allocate the

structure, zero it, and then populate it by calling `CPMLibGenerateKey()` or `CPMLibImportKeyInfo()`.

→ *`cipherInfo`*
A pointer to an `APCipherInfoStruct` that you can use to set the parameters of the encryption operation. If the CPM can't satisfy the requirements you specify in the structure, the operation will fail. If you want to use the default cipher settings, pass in a zero'd structure. When the function returns, the structure will be filled with information describing the operation.

→ *`inBuffer`*
A pointer to the data that you want to encrypt.

→ *`inBufferLength`*
The length of `inBuffer`, in bytes.

← *`outBuffer`*
A pointer to the buffer where the encrypted data will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the encrypted data.

↔ *`outBufferLength`*
You pass in the (allocated) length of `outBuffer`, in bytes. The function resets the argument to the amount of data that was actually copied into `outBuffer`. If the function returns `cpmErrBufTooSmall`, `outBufferLength` is set to the minimum buffer size that's needed to accommodate the encrypted data.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments This functions peforms block encryption. For stream encryption, see `CPMLibEncryptInit()`.

The `keyInfo` and `cipherInfo` must agree on the algorithm type, as specified in their respective `APAlgorithmEnum` fields.

If `outBuffer` isn't big enough, the function will fail and `outBufferLength` will return the "correct" output buffer size (i.e. large enough to accommodate the encrypted data) . When this happens, simply reallocate the output buffer and call `CPMLibEncrypt` again.

After you've encrypted the data, you must export the `keyInfo` and cipherInfo structures (see `CPMLibExportKeyInfo()` and `CPMLibExportCipherInfo()`) so they can be imported, later, by the data decryptor. Secure storage and transmission of the encryption key is th caller's responsibility.

To decrypt encrypted data, you call `CPMLibDecrypt()` or `CPMLibDecryptInit()`.

CPMLibEncryptFinal Function

Purpose Finalizes a stream encryption operation.

Declared In `CPMLib68kInterface.h, CPMLibARMInterface.h`

Prototype

```
Err CPMLibEncryptFinal ( UInt16 libRef,
   APKeyInfoType *keyInfo,
   APCipherInfoType *cipherInfo, UInt8 *inBuffer,
   UInt32 inBufferLength, UInt8 *outBuffer,
   UInt32 *outBufferLength)
```

Parameters

→ *libRef*
: CPM Library reference number (68k only).

→ *keyInfo*
: Key used to encrypt the data, as returned by `CPMLibGenerateKey()` or `CPMLibImportKeyInfo()`.

→ *cipherInfo*
: A pointer to the `APCipherInfoStruct` that was returned in the stream-initializing `CPMLibEncryptInit()` call.

→ *inBuffer*
: A pointer to the data that you want to encrypt. If you already supplied all the data through previous `CPMLibEncryptUpdate()` calls, pass `NULL`.

→ *inBufferLength*
: The length of `inBuffer`, in bytes. If `inBuffer` is `NULL`, pass 0.

← *outBuffer*
: A pointer to the buffer where the encrypted `inBuffer` data will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate the encrypted `inBuffer` data.

↔ *outBufferLength*
: You pass in the (allocated) length of `outBuffer`, in bytes. The function resets the argument to the amount of data that was actually copied into `outBuffer`. If the function returns `cpmErrBufTooSmall`, `outBufferLength` is set to the minimum buffer size that's needed to accommodate the encrypted data.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments This function (optionally) encrypts a final buffer of data and then closes the encryption stream that was initialized by `CPMLibEncryptInit()`.

After you've encrypted the data, you should export the `keyInfo` and `cipherInfo` structures (see `CPMLibExportKeyInfo()` and `CPMLibExportCipherInfo()`) so they can be imported, later, by the data decryptor. Secure storage and transmission of the encryption key is the caller's responsibility.

Note that `outBuffer` contains the encrypted `inBuffer` data only—it doesn't contain all the data that was encrypted by this stream. It's the callers responsibility to accumulate the data that was encrypted by previous, successive `CPMLibEncryptUpdate()` calls.

To decrypt encrypted data, you call `CPMLibDecrypt()` or `CPMLibDecryptInit()`.

CPMLibEncryptInit Function

Purpose Initializes a stream encryption session.

Declared In `CPMLib68kInterface.h, CPMLibARMInterface.h`

Prototype

```
Err CPMLibEncryptInit ( UInt16 libRef,
    APKeyInfoType *keyInfo,
    APCipherInfoType *cipherInfo )
```

Parameters

→ *libRef*
: CPM Library reference number (68k only).

→ *keyInfo*
Key used to encrypt the data, as returned by `CPMLibGenerateKey()` or `CPMLibImportKeyInfo()`.

→ *cipherInfo*
A pointer to an `APCipherInfoStruct` that you can use to set the parameters of the encryption operation. If the CPM can't satisfy the requirements you specify in the structure, the operation will fail. If you want to use the default cipher settings, pass in a zero'd structure. When the function returns, the structure will be filled with information describing the operation. The structure is used as a cookie in the subsequent `CPMLibEncryptUpdate()` and/or `CPMLibEncryptFinal()` functions.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments This function initializes a stream encryption operation. To feed data to the operation, you call `CPMLibEncryptUpdate()` followed by `CPMLibEncryptFinal()`. The "update" function is optional; the "final" function is mandatory. For block encryption, see `CPMLibEncrypt()`.

The `keyInfo` and `cipherInfo` must agree on the algorithm type, as specified in their respective `APAlgorithmEnum` fields.

To decrypt encrypted data, you call `CPMLibDecrypt()` or `CPMLibDecryptInit()`.

CPMLibEncryptUpdate Function

Purpose Feeds data to a stream encryption operation.

Declared In `CPMLib68kInterface.h, CPMLibARMInterface.h`

Prototype

```
Err CPMLibEncryptUpdate ( UInt16 libRef,
    APKeyInfoType *keyInfo,
    APCipherInfoType *cipherInfo, UInt8 *inBuffer,
    UInt32 inBufferLength, UInt8 *outBuffer,
    UInt32 *outBufferLength)
```

Parameters see `CPMLibEncryptFinal()`

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments This function feeds data into the stream encryption session that was started by `CPMLibEncryptInit()`. You can make any number of `CPMLibEncryptUpdate` calls while the encryption stream is open. When you've finished feeding data into the stream, you call `CPMLibEncryptFinal()`.

This function's arguments, return values, and behavior are nearly identical to `CPMLibEncryptFinal()` (which see for details). The only difference between them is that this function leaves the stream open, and `CPMLibEncryptFinal()` closes it.

CPMLibExportCipherInfo Function

Purpose Encodes a cipher into a form that can be cached. To reconstitute an exported cipher, pass it to `CPMLibImportCipherInfo()`.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibExportCipherInfo ( UInt16 libRef,
   APCipherInfoType *cipherInfo, UInt8 encoding,
   UInt8 *exportBuffer,
   UInt32 exportBufferLength )
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *cipherInfo*
Structure that represents the cipher that you want to export, as created and returned by `CPMLibGenerateKey()`, or as imported through `CPMLibImportKeyInfo()`.

→ *encoding*
Constant that specifies the type of encoding. One of `IMPORT_EXPORT_RAW`, `IMPORT_EXPORT_DER`, or `IMPORT_EXPORT_XML`. See `Export Encoding Constants` for details about these formats.

↔ *exportBuffer*
Buffer into which the function copies the encoded data. The buffer must be allocated by the caller. Point this argument to NULL if you're using the function to retrieve the size of the encoded data.

↔ *exportBufferLength*
You pass in the size of `exportBuffer` in bytes. The function returns (through this argument) the size that's required to accommodate the encoded data.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments You call this function twice: Once to get the size of the required exportBuffer, and then again (after allocating the buffer) to get the encoded buffer. See "Using the Export Functions" for more information and a free sample.

CPMLibExportHashInfo Function

Purpose Encodes an `APHashInfoStruct` into a form that can be cached. To reconstitute an exported hash info, pass it to `CPMLibImportHashInfo()`.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibExportKeyInfo ( UInt16 libRef,
   APHashInfoType *hashInfo, UInt8 encoding,
   UInt8 *exportBuffer,
   UInt32 exportBufferLength )
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *hashInfo*
Structure that you want to export.

→ *encoding*
Constant that specifies the type of encoding. One of `IMPORT_EXPORT_RAW`, `IMPORT_EXPORT_DER`, or `IMPORT_EXPORT_XML`. See `Export Encoding Constants` for details about these formats.

↔ *exportBuffer*
Buffer into which the function copies the encoded data. The buffer must be allocated by the caller. Point this argument to `NULL` if you're using the function to retrieve the size of the encoded data (see below

↔ *exportBufferLength*
You pass in the size of `exportBuffer` in bytes; the function returns (through this argument) the size that's required to accommodate the encoded data.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments You call this function twice: Once to get the size of the export buffer, and then again (after allocating the buffer) to retrieve the encoded data. See "Using the Export Functions" for more information and a free sample.

CPMLibExportKeyInfo Function

Purpose Encodes a `APKeyInfoStruct` into a form that can be cached. To reconstitute an exported key, pass it to `CPMLibImportKeyInfo()`.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibExportKeyInfo ( UInt16 libRef,
   APKeyInfoType *keyInfo, UInt8 encoding,
   UInt8 *exportBuffer,
   UInt32 exportBufferLength )
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *keyInfo*
Structure that you want to export.

→ *encoding*
Constant that specifies the type of encoding. One of `IMPORT_EXPORT_RAW`, `IMPORT_EXPORT_DER`, or `IMPORT_EXPORT_XML`. See `Export Encoding Constants` for details about these formats.

↔ *exportBuffer*
Buffer into which the function copies the encoded data. The buffer must be allocated by the caller. Point this argument to `NULL` if you're using the function to retrieve the size of the encoded data (see below

↔ *exportBufferLength*
You pass in the size of `exportBuffer` in bytes; the function returns (through this argument) the size that's required to accommodate the encoded data.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments You call this function twice: Once to get the size of the export buffer, and then again (after allocating the buffer) to retrieve the encoded data. See "Using the Export Functions" for more information and a free sample.

CPMLibExportVerifyInfo Function

Purpose Encodes a `APVerifyInfoStruct` into a form that can be cached. To reconstitute an exported key, pass it to `CPMLibImportVerifyInfo()`.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibExportKeyInfo ( UInt16 libRef,
   APVerifyInfoType *verifyInfo, UInt8 encoding,
   UInt8 *exportBuffer,
   UInt32 exportBufferLength )
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *keyInfo*
Structure that you want to export.

→ *encoding*
Constant that specifies the type of encoding. One of `IMPORT_EXPORT_RAW`, `IMPORT_EXPORT_DER`, or `IMPORT_EXPORT_XML`. See `Export Encoding Constants` for details about these formats.

↔ *exportBuffer*
Buffer into which the function copies the encoded data. The buffer must be allocated by the caller. Point this argument to `NULL` if you're using the function to retrieve the size of the encoded data (see below

↔ *exportBufferLength*
You pass in the size of `exportBuffer` in bytes; the function returns (through this argument) the size that's required to accommodate the encoded data.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments You call this function twice: Once to get the size of the export buffer, and then again (after allocating the buffer) to retrieve the encoded data. See "Using the Export Functions" for more information and a free sample.

CPMLibGenerateKey Function

Purpose Generates a new symmetric key.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibGenerateKey ( UInt16 libRef,
   UInt8 *seedData, UInt32 seedLength,
   APKeyInfoType *keyInfo )
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *seedData*
Optional data that's used to seed the key generator. Because PalmOS Garnet doesn't currently support key "derivation" (identical key generation based on identical seeds) the seed data is, in essence, a no-op, and can be a pointer to 0. However, you may want to supply (and cache) unique seed data today in anticipation of tomorrow's derivation functionality.

→ *seedLength*
Length of `seedData`, in bytes. (Pass 0 if `*seedData` is 0.)

↔ *keyInfo*
A pointer to a `APKeyInfoStruct` that will contain the generated key. The `keyInfo` structure is allocated and owned by the caller. You can specify the desired provider, key-generation scheme, and so on, by setting the `APKeyInfoType` fields. Or, to retrieve a default key, zero the structure before you pass it in.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments The `APKeyInfoStruct` that's populated by this function can be used in subsequent encryption, decryption, and verification operations.

When you're finished using the `APKeyInfoStruct`, you must release it through `CPMLibReleaseKeyInfo()`.

CPMLibGetInfo Function

Purpose Retrieves information about the CPM library.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibLibGetInfo ( UInt16 libRef,
    CPMInfoType *info)
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *info*
Pointer to a `CPMInfoStruct` that returns the information about the library. See `CPMInfoStruct` for a description.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

CPMLibGetProviderInfo Function

Purpose Retrieves information about a specific provider.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibGetProviderInfo ( UInt16 libRef,
    UInt32 providerID,
    APProviderInfoType *providerInfo)
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *providerID*
ID number that identifies the provider.

← *providerInfo*
Structure into which the function places provider information. The structure is allocated and freed by the caller. The function doesn't clear inapplicable fields on the way out.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

CPMLibHash Function

Purpose Hashes a block of data.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibHash ( UInt16 libRef, APHashEnum type,
   APHashInfoType *hashInfo, UInt8 *inBuffer,
   UInt32 inBufferLength, UInt8 *outBuffer,
   UInt32 *outBufferLength )
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *type*
A constant that represents the hashing algorithm that will be used to create the message digest. See Hashing Algorithm Constants for a list of constants. If you want the default algorithm, use `apHashTypeUnspecified`.

→ *hashInfo*
A pointer to an `APHashInfoStruct` that you can use to set the parameters of the hashing operation. If the CPM can't satisfy the requirements you specify in the structure, the operation will fail. If you want to use the default settings, pass in a zero'd structure. When the function returns, the structure will be filled with information describing the operation.

→ *inBuffer*
A pointer to the data that you want to hash.

→ *inBufferLength*
The length of `inBuffer`, in bytes. If `inBuffer` is `NULL`, pass 0.

← *outBuffer*
A pointer to the buffer where the hashed `inBuffer` data will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the hashed data.

↔ *outBufferLength*
You pass in the (allocated) length of `outBuffer`, in bytes. The function resets the argument to the amount of data that was actually copied into `outBuffer.` If the function returns `cpmErrBufTooSmall`, `outBufferLength` is set to the minimum buffer size that's needed to accommodate the hashed data..

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments This function performs a block hash operation. For stream hashing, use `CPMLibHashInit()`.

CPMLibHashFinal Function

Purpose Finalizes a hash session and returns the hashed data.

Declared In `CPMLib.h`

Prototype

```
Err CPMLibHashFinal ( UInt16 libRef,
   APHashInfoType *hashInfo, UInt8 *inBuffer,
   UInt32 inBufferLength, UInt8 *outBuffer,
   UInt32 *outBufferLength )
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *hashInfo*
A pointer to the `APHashInfoStruct` that was returned by `CPMLibHashInit()`.

→ *inBuffer*
A pointer to the data that you want to hash. If you already supplied all the data through previous `CPMLibHashUpdate()` calls, pass `NULL`.

→ *inBufferLength*
The length of `inBuffer`, in bytes. If `inBuffer` is `NULL`, pass 0.

← *outBuffer*
A pointer to the buffer where all the data that has been hashed by this stream will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the hashed data.

↔ *outBufferLength*
You pass in the (allocated) length of `outBuffer`, in bytes. The function resets the argument to the amount of data that was actually copied into `outBuffer`. If the function returns `cpmErrBufTooSmall`, `outBufferLength` is set to the minimum buffer size that's needed to accommodate the hashed data.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments This function returns all the data that was hashed by the hash stream, and then closes the stream. It follows an initial call to `CPMLibHashInit()` and some number of calls to `CPMLibHashUpdate()`.

CPMLibHashInit Function

Purpose Initiates a streaming hash operation.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibHash ( UInt16 libRef,
   APHashInfoType *hashInfo )
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *hashInfo*
A pointer to an `APHashInfoStruct` that you can use to set the parameters of the hash operation. If the CPM can't satisfy the requirements you specify in the structure, the operation will fail. If you want to use the default settings, pass in a zero'd structure. When the function returns, the structure will be filled with information describing the operation.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments This function initializes a streaming hash operation. To feed data to the stream, you call `CPMLibHashUpdate()` followed by `CPMLibHashFinal()`. The "update" function is optional; the "final" function is mandatory. For a block hash, see `CPMLibHash()`.

CPMLibHashUpdate Function

Purpose Sends data to streaming hash operation.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibHashUpdate ( UInt16 libRef,
   APHashInfoType *hashInfo, UInt8 *inBuffer,
   UInt32 inBufferLength)
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *hashInfo*
A pointer to the `APHashInfoStruct` that was returned by `CPMLibHashInit()`.

→ *inBuffer*
A pointer to the data that you want to hash.

→ *inBufferLength*
The length of `inBuffer`, in bytes.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments This function feeds data into the streaming hash session that was started by `CPMLibHashInit()`. You can make any number of `CPMLibHashUpdate` calls while the hash stream is open. When you've finished feeding data into the stream, you call `CPMLibHashFinal()`.

Note that this function doesn't return any hashed data. All the data that's hashed by the stream is returned through the `CPMLibHashFinal()` call.

CPMLibImportCipherInfo Function

Purpose Imports a previously-exported cipher info structure so that it can be used in subsequent operations.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibImportCipherInfo ( UInt16 libRef,
    UInt8 encoding, UInt8 *importData,
    UInt32 importDataLength,
    APCipherInfoType *cipherInfo )
```

Parameters

→ *libRef*
: CPM Library reference number (68k only).

→ *encoding*
: Constant that specifies the encoding type that was used to export the key. One of `IMPORT_EXPORT_RAW`, `IMPORT_EXPORT_DER`, or `IMPORT_EXPORT_XML`. See `Export Encoding Constants` for details about these formats.

→ *importData*
: The encoded data.

→ *importDataLength*
: The length of `importData`, in bytes.

← *cipherInfo*
: `APCipherInfoStruct` that returns the imported key. The structure must be allocated before it's passed in.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

CPMLibImportHashInfo Function

Purpose Imports a previously-exported hash info structure so that it can be used in subsequent operations.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibImportHashInfo ( UInt16 libRef,
    UInt8 encoding, UInt8 *importData,
    UInt32 importDataLength,
    APHashInfoType *keyInfo )
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *encoding*
Constant that specifies the encoding type that was used to export the key. One of `IMPORT_EXPORT_RAW`, `IMPORT_EXPORT_DER`, or `IMPORT_EXPORT_XML`. See `Export Encoding Constants` for details about these formats.

→ *importData*
The encoded data.

→ *importDataLength*
The length of `importData`, in bytes.

← *hashInfo*
`APHashInfoStruct` that returns the imported hash info. The structure must be allocated before it's passed in.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

CPMLibImportKeyInfo Function

Purpose Imports a previously-exported key so that it (the key) can be used in subsequent operations.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibImportKeyInfo ( UInt16 libRef,
    UInt8 encoding, UInt8 *importData,
    UInt32 importDataLength,
    APKeyInfoType *keyInfo )
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *encoding*
Constant that specifies the encoding type that was used to export the key. One of `IMPORT_EXPORT_RAW`, `IMPORT_EXPORT_DER`, or `IMPORT_EXPORT_XML`. See `Export Encoding Constants` for details about these formats.

→ *importData*
The encoded data.

→ *importDataLength*
The length of `importData`, in bytes.

← *keyInfo*
`APKeyInfoStruct` that returns the imported key. The structure must be allocated before it's passed in.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

CPMLibImportVerifyInfo Function

Purpose Imports a previously-exported verify info structure so that it can be used in subsequent operations.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibImportVerifyInfo ( UInt16 libRef,
    UInt8 encoding, UInt8 *importData,
    UInt32 importDataLength,
    APVerifyInfoType *keyInfo )
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *encoding*
Constant that specifies the encoding type that was used to export the key. One of `IMPORT_EXPORT_RAW`, `IMPORT_EXPORT_DER`, or `IMPORT_EXPORT_XML`. See `Export Encoding Constants` for details about these formats.

→ *importData*
The encoded data.

→ *importDataLength*
The length of `importData`, in bytes.

← *verifyInfo*
`APVerifyInfoStruct` that returns the imported verify info. The structure must be allocated before it's passed in.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

CPMLibReleaseCipherInfo Function

Purpose Releases a `APCipherInfoStruct`, allowing you to free it.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibReleaseCipherInfo ( UInt16 libRef,
    APCipherInfoType *cipherInfo )
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *cipherInfo*
A pointer to a `APCipherInfoStruct` that you want to release.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

CPMLibReleaseHashInfo Function

Purpose Releases a `APKeyInfoStruct`, allowing you to free it.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibReleaseHashInfo ( UInt16 libRef,
   APHashInfoType *hashInfo )
```

Parameters → *libRef*
CPM Library reference number (68k only).

→ *hashInfo*
A pointer to a `APHashInfoStruct` that you want to release.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

CPMLibReleaseKeyInfo Function

Purpose Releases a `APKeyInfoStruct`, allowing you to free it.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibReleaseKeyInfo ( UInt16 libRef,
   APKeyInfoType *keyInfo )
```

Parameters → *libRef*
CPM Library reference number (68k only).

→ *keyInfo*
A pointer to the `APKeyInfoStruct` that you want to release.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

CPMLibReleaseVerifyInfo Function

Purpose Releases a `APVerifyInfoStruct`, allowing you to free it.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibReleaseVerifyInfo ( UInt16 libRef,
   APVerifyInfoType *verifyInfo )
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *verifyInfo*
A pointer to a `APVerifyInfoStruct` that you want to release.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

CPMLibVerify Function

Purpose Verifies a message.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibVerify ( UInt16 libRef,
   APKeyInfoType *keyInfo,
   APVerifyInfoType *verifyInfo, UInt8 *inBuffer,
   UInt32 inBufferLength, UInt8 *outBuffer,
   UInt32 *outBufferLength, UInt8 *signature,
   UInt32 signatureLength,
   VerifyResultType *verifyResult )
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *keyInfo*
An `APKeyInfoStruct` that represents the certificate's encryption key. Extracting the key data from the certificate and constructing the `APKeyInfoStruct` (through `CPMLibImportKeyInfo()`) is the caller's responsibility.

→ *verifyInfo*
A pointer to an `APVerifyInfoStruct` that specifies the hash and cipher operations that should be performed during verification. This information is embedded as `APHashInfoStruct` and `APCipherInfoStruct`

structures. If you want to use the default operations, allocate and zero the embedded structures. When the function returns, the structures will be populated with information describing the operations that were used.

→ *inBuffer*
A pointer to the message data.

→ *inBufferLength*
The length of `inBuffer`, in bytes.

← *outBuffer*
A pointer to the buffer where the decrypted signature will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the decrypted data. If you don't care about the signature, pass `NULL`.

↔ *outBufferLength*
If `outBuffer` is `NULL`, set this to 0. Otherwise, you pass in the (allocated) length of `outBuffer`, in bytes. The function resets the argument to the amount of data that was actually copied into `outBuffer.` If the function returns `cpmErrBufTooSmall`, `outBufferLength` is set to the minimum buffer size that's needed to accommodate the verified data.

→ *signature*
A pointer to the message's signature. Extracting the signature from the message is the caller's responsibility

→ *signatureLength*
The length of the signature, in bytes.

← *verifyResult*
An integer point that returns, by reference, the result of the verification. 0 means the message was verified; non-zero means is wasn't. The meaning of a non-zero return is defined by the algorithm provider.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments This function performs a block verifcation operation. For stream verification, use `CPMLibVerifyInit()`.

Keep in mind that a direct return of `errNone` doesn't mean that the message has been verified. The verification status is returned by reference in `verifyResult`.

CPMLibVerifyFinal Function

Purpose Finalizes a verify session.

Declared In `CPMLib.h`

Prototype

```
Err CPMLibVerifyFinal ( UInt16 libRef,
   APKeyInfoType *keyInfo,
   APVerifyInfoType *verifyInfo, UInt8 *inBuffer,
   UInt32 inBufferLength, UInt8 *outBuffer,
   UInt32 *outBufferLength, UInt8 *signature,
   UInt32 signatureLength,
   VerifyResultType *verifyResult )
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *verifyInfo*
A pointer to the `APVerifyInfoStruct` that was returned by `CPMLibVerifyInit()`.

→ *inBuffer*
A pointer to the final buffer of message data, or `NULL` if there's no more data..

→ *inBufferLength*
The length of `inBuffer`, in bytes.

← *outBuffer*
A pointer to the buffer where the decrypted signature will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the decrypted data. If you don't care about the signature, pass `NULL`.

↔ *outBufferLength*
If `outBuffer` is `NULL`, set this to 0. Otherwise, you pass in the (allocated) length of `outBuffer`, in bytes. The function resets the argument to the amount of data that was actually copied into `outBuffer.` If the function returns `cpmErrBufTooSmall`, `outBufferLength` is set to the minimum buffer size that's needed to accommodate the verified data.

→ *signature*
A pointer to the message's signature. Extracting the signature from the message is the caller's responsibility

→ *signatureLength*
The length of the signature, in bytes.

← *verifyResult*
An integer point that returns, by reference, the result of the verification. 0 means the message was verified; non-zero means is wasn't. The meaning of a non-zero return is defined by the algorithm provider.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments This function feeds a final (optional) buffer of message into a verify stream that was previously initialized by `CPMLibVerifyInit()` and augmented through successive calls to `CPMLibVerifyUpdate()`. The entire message is then verified, the results of the verification are returned in `verifyResult`, and the stream is closed.

CPMLibVerifyInit Function

Purpose Initiates a streaming verify operation.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibVerify ( UInt16 libRef,
   APKeyInfoType *keyInfo,
   APVerifyInfoType *verifyInfo )
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *keyInfo*
An `APKeyInfoStruct` that represents the certificate's encryption key. Extracting the key data from the certificate and constructing the `APKeyInfoStruct` (through `CPMLibImportKeyInfo()`) is the caller's responsibility.

→ *verifyInfo*
A pointer to an `APVerifyInfoStruct` that specifies the hash and cipher operations that should be performed during verification. This information is embedded as `APHashInfoStruct` and `APCipherInfoStruct` structures. If you want to use the default operations, allocate and zero the embedded structures. When the function

returns, the structures will be populated with information describing the operations that were used.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments This function initializes a streaming hash operation. To feed data to the stream, you call `CPMLibVerifyUpdate()` followed by `CPMLibVerifyFinal()`. The "update" function is optional; the "final" function is mandatory. For a block hash, see `CPMLibVerify()`.

CPMLibVerifyUpdate Function

Purpose Sends message data to a streaming verify operation.

Declared In `CPMLib68KInterface.h, CPMLibArmInterface.h`

Prototype

```
Err CPMLibVerifyUpdate ( UInt16 libRef,
   APKeyInfoType *keyInfo,
   APVerifyInfoType *verifyInfo )
   UInt8 *inBuffer, UInt32 inBufferLength)
```

Parameters

→ *libRef*
CPM Library reference number (68k only).

→ *keyInfo*
The `APKeyInfoStruct` that was used in the `CPMLibVerifyInit()` call.

→ *verifyInfo*
A pointer to the `APVerifyInfoStruct` that was returned by `CPMLibVerifyInit()`.

→ *inBuffer*
A pointer to the message data.

→ *inBufferLength*
The length of `inBuffer`, in bytes.

Returns The function returns `errNone` upon success. For other error codes, see CPM Error Codes.

Comments This function feeds message data into the verify stream that was intialized by `CPMLibVerifyInit()`. When you're finished feeding data into the stream, call `CPMLibVerifyFinal()` to close the stream and return the verification results.

85

SMS Exchange Library

This chapter describes the SMS Exchange Library API declared in the header file `SmsLib.h`. It discusses the following topics:

- SMS Exchange Library Data Structures
- SMS Exchange Library Constants

You interact with the SMS Exchange Library using the Exchange Manager APIs described in Chapter 62, "Exchange Manager," on page 1355 of this book. For further information on using Exchange Manager, see "Object Exchange" on page 1 of the of the *Palm OS Programmer's Companion*, vol. II, *Communications*.

SMS Exchange Library Data Structures

SmsParamsType Struct

Purpose

The `SmsParamsType` structure identifies information specific to the SMS Exchange Library. The `socketRef` field of the `ExgSocketType` structure is set to this structure when you send or receive data using the SMS Exchange Library. You only need to create this structure and assign it to the `socketRef` field if you have an SMS message to send and want to use non-default values for some of the fields; otherwise, the SMS Exchange Library creates this structure for you and provides default values.

Prototype

```
typedef struct SmsParamsTag {
  UInt32 creator;
  UInt16 smsID;
  Char *extension;
  Char *mimeTypes;
  UInt32 appCreator;
```

```
  UInt8 dataCodingScheme;
  UInt8 networkType;
  UInt8 dataType;
  UInt16 nbsDestPort;
  UInt16 nbsSrcPort;
  union {
    SmsSendParamsType send;
    SmsReceiveParamsType receive;
    SmsReportParamsType report;
  } data;
} SmsParamsType, *SmsParamsPtr;
```

Fields

`creator`
: Creator ID of the SMS Exchange Library. Always set this to `sysFileCSmsLib`.

`smsID`
: The ID of the message that was sent. Do not set this field directly; the SMS Exchange Library should set it.

`extension`
: If the SMS message has an attachment, this field specifies the attachment name. Do not set this field directly; the SMS Exchange Library sets it if necessary. See the description of the `appCreator` field for details.

`mimeTypes`
: If the SMS message has an attachment, this field specifies the MIME type of the attachment. Do not set this field directly; the SMS Exchange Library sets it if necessary. See the description of the `appCreator` field for details.

`appCreator`
: The creator ID of the target application for the attachment to the SMS message. Do not set this field directly; the SMS Exchange Library sets it if necessary.

  When the SMS Exchange Library receives a message with an attachment, it unwraps the message and attempts to deliver the attachment directly to an application that is registered to receive it. If no application is registered to receive unwrapped attachments of that type, the SMS Exchange Library sends the entire SMS message, and it sets the `extension`, `mimeTypes`, and `appCreator` fields in this structure. The SMS application can use this information to

have the Exchange Manager deliver the attachment to the appropriate application using the Local Exchange Library.

`dataCodingScheme`
: The data encoding scheme that the message uses. See SMS Data Coding Scheme Constants.

`networkType`
: Indicates the type of advanced parameters. See SMS Network Type Constants.

`dataType`
: Identifies the type of message being received, such as multipart or return receipt. See SMS Message Type Constants.

`nbsDestPort`
: The Narrow Band Socket (NBS) port on which you want the data sent. The SMS Exchange Library sets this for you if you leave it blank. When data is being received, this field is set to the NBS port on which the data was received.

`nbsSrcPort`
: The NBS port on which you want the data sent. You should set this field to the same value as `nbsDestPort`. If you leave it blank, the SMS Exchange Library provides a value for you.

`data`
: The data being sent or received. See `SmsReceiveParamsType`, `SmsReportParamsType`, and `SMSSendParamsType`.

Compatibility This structure is only defined if 4.0 New Feature Set is present.

SmsPrefType Struct

Purpose The `SmsPrefType` structure defines the SMS Exchange Library preferences for sending and receiving SMS messages. Applications can use the `ExgControl()` function to get, set, or display these preferences to the user.

Prototype

```
typedef struct SmsPrefTag {
  UInt32 validity;
  UInt16 warnOver;
  Boolean leave;
  Boolean report;
  Boolean autoSMSC;
  Char smscNumber[kSmsMaxPhoneSize];
} SmsPrefType, *SmsPrefPtr;
```

Fields

`validity`
: The number of seconds before the message expires. If the message cannot be delivered to the recipient, the service center repeatedly attempts to deliver the message until it expires. The default is one hour.

`warnOver`
: The number of parts a user can send without confirmation. If the user attempts to send a message with more than this number of parts, an alert is displayed, and the user can choose to send the message anyway. The default is 3 parts. (If the user attempts to send a message with more than 3 parts, an alert is displayed.)

`leave`
: If `true`, any incoming messages retrieved from a phone remain on the phone as well. If `false`, the messages are deleted from the phone's inbox.

`report`
: If `true`, the user receives confirmation that an SMS message was delivered.

`autoSMSC`
: If `true`, don't use the value stored in the `smscNumber` field.

`smscNumber`
: The message center to be used. If `NULL` or the empty string, the SMS message center set by the phone is used.

Compatibility

This structure is only defined if 4.0 New Feature Set is present.

SmsReceiveCDMAParamsType Struct

Purpose

The `SmsReceiveParamsType` includes an `SmsReceiveCDMAParamsType` structure for CDMA messages.

Prototype

```
typedef struct SmsReceiveCDMAParamsTag {
  UInt8 messageType;
  TelSmsDateTimeType validityPeriod;
  UInt8 priority;
  UInt8 privacy;
  Boolean alertOnDeliveryRequest;
  Boolean manualAckRequest;
  UInt8 voiceMessageNumber;
  UInt8 languageIndicator;
  Char *callbackNumberAddress;
} SmsReceiveCDMAParamsType,
*SmsReceiveCDMAParamsPtr;
```

Fields

`messageType`
: The type of the message. This is one of the SMS Message Type Constants constants defined in `TelephonyMgr.h`.

`validityPeriod`
: The amount of time for which the message is valid. See `TelSmsDateTimeType`. The default is set according to the SMS preferences.

`priority`
: The message priority. This must be one of the SMS Message Urgency Constants defined in `TelephonyMgr.h`.

`privacy`
: The privacy type of the message. This must be one of the SMS Message Privacy Constants defined in `TelephonyMgr.h`.

`alertOnDeliveryRequest`
: `true` if the user is to be alerted upon delivery of this message, and `false` if not.

`manualAckRequest`
: `true` if a confirmation is requested from the recipient, and `false` if not.

`voiceMessageNumber`
: The number of new messages in your voice mail.

`languageIndicator`
: Reserved for future use.

`callbackNumberAddress`
: The callback number to which confirmations are to be sent.

Compatibility This structure is only defined if 4.0 New Feature Set is present.

SmsReceiveGSMParamsType Struct

Purpose The `SmsReceiveParamsType` includes an `SmsReceiveGSMParamsType` structure for GSM messages.

Prototype

```
typedef struct SmsReceiveGSMParamsTag {
  UInt16 protocolId;
  Char *serviceCenterNumber;
  Boolean replyPath;
} SmsReceiveGSMParamsType,
*SmsReceiveGSMParamsPtr;
```

Fields

`protocolId`
: Reserved for future use.

`serviceCenterNumber`
: The SMS service center that must be used to send a reply. If `NULL`, the service center specified in the preferences is used.

`replyPath`
: If `true`, replies must be made through the SMS service center specified by `serviceCenterNumber`.

Compatibility This structure is only defined if 4.0 New Feature Set is present.

SmsReceiveParamsType Struct

Purpose The `SmsReceiveParamsType` structure is used as the `data` field for the `SmsParamsType` structure when the SMS Exchange Library has received data. The SMS Exchange Library always supplies the values for these fields.

Prototype

```
typedef struct SmsReceiveParamsTag {
  UInt32 timeStamp;
  Char *originatingAddress;
  UInt8 leaveOnPhone:1;
  UInt8 forceSlotMode:1;
  UInt8 reserved:6;
  UInt16 index;
  Boolean otherToReceive;
  Boolean reportDeliveryIndicator;
```

```
  union   {
    SmsReceiveGSMParamsType gsm;
    SmsReceiveCDMAParamsType cdma;
    SmsReceiveTDMAParamsType tdma;
  } protocol;
} SmsReceiveParamsType, *SmsReceiveParamsPtr;
```

Fields

`timeStamp`
: The time at which the message was delivered, given as the number of seconds since January 1, 1904.

`originatingAddress`
: The number from which the message was received.

`leaveOnPhone`
: If `true`, messages received on the phone are not deleted from the phone's inbox. If not specified, this is set according to the system preferences.

`forceSlotMode`
: If `true`, use slot mode parsing. If `false`, use block mode parsing. The default is block mode.

`reserved`
: Reserved for future use.

`index`
: Location where the message is stored on the mobile phone.

`otherToReceive`
: If `true`, there are more messages to be received from the service center.

`reportDeliveryIndicator`
: If `true`, the sender has requested confirmation. The recipient of the message does not send the confirmation; the SMS service center does.

`protocol`
: Values specific to the protocol used to send the message. Currently, only GSM is supported.

Compatibility

This structure is only defined if 4.0 New Feature Set is present.

SmsReceiveTDMAParamsType Typedef

Purpose The `SmsReceiveParamsType` includes an `SmsReceiveTDMAParamsType` structure for TDMA messages. This structure is currently the same as the `SmsReceiveCDMAParamsType` structure.

Prototype

```
typedef SmsReceiveCDMAParamsType
SmsReceiveTDMAParamsType,
*SmsReceiveTDMAParamsPtr;
```

Compatibility This structure is only defined if 4.0 New Feature Set is present.

SmsReportParamsType Struct

Purpose The `SMSReportParamsType` structure is used as the `data` field for the `SmsParamsType` structure when the SMS Exchange Library has received a delivery confirmation. The SMS Exchange Library always sets the values for these fields.

Prototype

```
typedef struct SmsReportParamsTag {
  UInt32 timeStamp;
  UInt16 index;
  UInt8 reportType;
  UInt8 report;
  Char *originatingAddress;
} SmsReportParamsType, *SmsReportParamsPtr;
```

Fields

`timeStamp`
: The date and time at which the message was delivered, given as the number of seconds since January 1, 1904.

`index`
: Location where the message is stored on the mobile phone.

`reportType`
: One of the Delivery Report Type constants defined in `TelephonyMgr.h`.

`report`
: One of the Delivery Status Report constants defined in `TelephonyMgr.h`.

`originatingAddress`
: Phone number to which the message was sent.

Compatibility This structure is only defined if 4.0 New Feature Set is present.

SmsSendCDMAParamsType Struct

Purpose The `SMSSendParamsType` includes an `SmsSendCDMAParamsType` structure for CDMA messages.

Prototype

```
typedef struct SmsSendCDMAParamsTag {
  UInt8 messageType;
  TelSmsDateTimeType deferredDate;
  UInt8 priority;
  UInt8 privacy;
  UInt8 alertOnDelivery:1;
  UInt8 manualAckRequest:1;
  UInt8 reserved:6;
  Char *callbackNumber;
} SmsSendCDMAParamsType, *SmsSendCDMAParamsPtr;
```

Fields

`messageType`
: The type of the message. This is one of the SMS Message Type Constants constants defined in `TelephonyMgr.h`.

`deferredDate`
: Not used.

`priority`
: The message priority. This must be one of the SMS Message Urgency Constants.

`privacy`
: The privacy type of the message. This must be one of the SMS Message Privacy Constants.

`alertOnDelivery`
: `true` if the user is to be alerted upon delivery of this message, and `false` if not.

`manualAckRequest`
: `true` if a confirmation is requested from the recipient, and `false` if not.

`reserved`
: Reserved for future use.

`callbackNumber`
: Number to which the confirmation should be sent.

Compatibility This structure is only defined if 4.0 New Feature Set is present.

SmsSendGSMParamsType Struct

Purpose The `SMSSendParamsType` includes an `SmsSendGSMParamsType` structure for GSM messages.

Prototype

```
typedef struct SmsSendGSMParamsTag {
  UInt16 protocolId;
  Char *serviceCenterNumber;
  Boolean rejectDuplicated;
  Boolean replyPath;
} SmsSendGSMParamsType, *SmsSendGSMParamsPtr;
```

Fields

`protocolId`
: Reserved for future use.

`serviceCenterNumber`
: The message center to be used. If not specified, the service center is set according to the system preferences.

`rejectDuplicated`
: If `true`, the service center rejects messages that have the same message ID, destination address, and originating address as a previously submitted message.

`replyPath`
: If `true`, the service center that delivers the message is requested to provide information about itself to the recipient so that replies are made through the same service center.

Compatibility This structure is only defined if 4.0 New Feature Set is present.

SMSSendParamsType Struct

Purpose The `SMSSendParamsType` structure is used as the `data` field for the `SmsParamsType` structure when the SMS Exchange Library is sending data.

Prototype

```
typedef struct SmsSendParamsTag {
  TelSmsDateTimeType validityPeriod;
  Char *destinationAddress;
  UInt8 networkDeliveryRequested:1;
  UInt8 ignoreDefaultValue:1;
```

```
  UInt8 reserved:6;
  UInt16 partCount;
  UInt16 lastPart;
  UInt8 converter;
  union   {
    SmsSendGSMParamsType gsm;
    SmsSendCDMAParamsType cdma;
    SmsSendTDMAParamsType tdma;
  } protocol;
} SmsSendParamsType, *SmsSendParamsPtr;
```

Fields

`validityPeriod`
: The amount of time for which the message is valid. See `TelSmsDateTimeType`. The default is set according to the SMS preferences.

`destinationAddress`
: A buffer that contains the phone number of the message recipient. If no phone number is supplied, the user is prompted for the phone number.

`networkDeliveryRequested`
: If `true`, the SMS service center sends a delivery confirmation. The default is set according to the SMS preferences. The SMS Exchange Library disables this field for multipart messages.

`ignoreDefaultValue`
: If `false`, the validity period, network delivery requested, and SMS center specified in the preferences are used regardless of the values supplied in this structure. If `true`, the values supplied in this structure are used.

`reserved`
: Reserved for future use.

`partCount`
: The number of parts in the message. 0 means that the message is not a multipart message. If NBS is used to send the message, it determines the number of parts.

`lastPart`
: The last part of a multipart message that was successfully sent.

`converter`
: The header added to the data to specify how it is converted.

`protocol`
: Data specific to the protocol used to send the message. Currently, only GSM is supported.

Compatibility This structure is only defined if 4.0 New Feature Set is present.

SmsSendTDMAParamsType Typedef

Purpose The `SMSSendParamsType` includes an `SmsSendTDMAParamsType` structure for TDMA messages. This structure is currently the same as the `SmsSendCDMAParamsType` structure.

Prototype

```
typedef SmsSendCDMAParamsType
SmsSendTDMAParamsType, *SmsSendTDMAParamsPtr;
```

Compatibility This structure is only defined if 4.0 New Feature Set is present.

SMS Exchange Library Constants

SMS Control Constants

The SMS control constants are passed as the operation parameter to the `ExgControl()` function. The `ExgControl` function is a way to communicate directly with the SMS Exchange Library. The following table lists the operation constant, the type of data that should be passed as the `valueP` parameter to `ExgControl()`, and what the SMS Exchange Library does in response.

Table 85.1 ExgControl operations for SMS library

Operation	value Data Type	Description
`exgLibSmsPrefGetOp`	`SmsPrefType`	Returns a pointer to the SMS Exchange Libraries preferences in `valueP`, creating the preferences and setting them to the default values if they do not exist.
`exgLibSmsPrefGet DefaultOp`	`SmsPrefType`	Returns the default values for the SMS Exchange Library preferences.
`exgLibSmsPrefSetOp`	`SmsPrefType`	Sets the SMS Exchange Library preferences to the values passed in `valueP`.
`exgLibSmsPref DisplayOp`	`kSmsNetworkAuto` or `kSmsNetworkGSM`. Input only.	Display a form that allows the user to set the SMS preferences.
`exgLibSms IncompleteGetCountOp`	`UInt16`. Output only.	Get the number of incomplete messages currently stored in the SMS Exchange Library. The library stores message parts as it receives them. When it has received all of the parts, it reassembles the message and delivers it. This operation tells how many messages are currently under assembly.
`exgLibSms IncompleteDeleteOp`	`UInt16`. Input only.	Delete the incomplete message with the ID passed in `valueP`. Pass -1 to delete all incomplete messages.

Compatibility These constants are only defined if 4.0 New Feature Set is present.

SMS Data Coding Scheme Constants

The SMS data coding scheme constants describe the coding scheme used for SMS data. These values are used as the `dataCodingScheme` parameter of the `SmsParamsType` structure.

Constant	Value	Description
`kSmsRowDataEncoding`	0	8-bit encoding scheme. This is the default.
`kSmsTextEncoding`	1	7-bit encoding scheme.

Compatibility These constants are only defined if 4.0 New Feature Set is present.

SMS Network Type Constants

The SMS network type constants identify the type of network being used for SMS messages. Currently, only the GSM network is supported.

Constant	Value	Description
`kSmsNetworkAuto`	`-1`	The network is set by the phone. This is the default.
`kSmsNetworkCDMA`	`kTelNwkCDMA`	A CDMA network. Currently not supported.
`kSmsNetworkGSM`	`kTelNwkGSM`	A GSM network.
`kSmsNetworkTDMA`	`kTelNwkTDMA`	A TDMA network. Currently not supported.
`kSmsNetworkPDC`	`kTelNwkPDC`	A PDC network. Currently not supported.

Compatibility These constants are only defined if 4.0 New Feature Set is present.

SMS Message Type Constants

The SMS message type constants identify the type of message being sent. They are used as the `dataType` field of the `SmsParamsType` structure.

Constant	Value	Description
`kSmsMessageType`	0	Standard SMS message of no more than 160 bytes. This is the default.
`kSmsIncompleteType`	1	A part of a multipart SMS message.
`kSmsReportType`	2	A confirmation, indicating that an SMS message was successfully sent.

Compatibility These constants are only defined if 4.0 New Feature Set is present.

SMS Converter Constants

The SMS converter constants identify the header information added to an SMS message.

Constant	Value	Description
`kSmsNBSConverter`	0	An NBS header is added to the message. This is the default.
`kSmsNoConverter`	1	No header is added to the message.

Compatibility These constants are only defined if 4.0 New Feature Set is present.

86

SSL Library

This chapter contains reference documentation for the APIs defined in `SslLib.h`. The contents of this chapter are organized into the following sections:

SSL Library Structures and Types 2189
SSL Library Constants 2202
SSL Library Functions and Macros 2219
Application-Defined Functions 2241

The header file `SslLib.h` declares the API that this chapter describes.

For information on making use of the APIs documented in this chapter, see Chapter 8, "Secure Sockets Layer (SSL)," on page 203 of *Palm OS Programmer's Companion*, vol. II, *Communications*. Much of what you do when working with the SSL Library involves working with attributes; reference documentation for the macros that you use to set and get attribute values can be found in Chapter 87, "SSL Library Macros," on page 2251.

SSL Library Structures and Types

SslAttribute Typedef

Purpose Container for an SslLib or SslContext attribute value.

Declared In `SslLib.h`

Prototype

```
typedef UInt32 SslAttribute;
```

Comments Attribute values are listed in "SSL Library Macro Constants" on page 2251. Descriptions of each attribute can be found under "Attributes" on page 206.

SslCallback Struct

Purpose Structure used when the SSL library transfers control back to the application via a callback function.

Declared In `SslLib.h`

Prototype

```
struct SslCallback_st {
   void *reserved;
   SslCallbackFunc callback;
   void *data;
   SslContext *ssl;
}
typedef struct SslCallback_st SslCallback
```

Fields

`reserved`
: Reserved for internal use by SslLib.

`callback`
: A function pointer of the function to be called. See `SslCallbackFunc()`.

`data`
: Can be set to a value by the application, and will remain unchanged by SslLib. This value will then be available to the callback function. Use this field to communicate context information from the application to the callback. An example of its use is in a 'diagnostic' callback, in which the application could use this field to provide a handle to the logging routines to be used.

`ssl`
: SslLib sets this field to be the `SslContext` if the callback is related to an SslContext. If it is not (due to being related to a SslLib), it is set to `NULL`.

Comments This structure is used when the SSL library transfers control back to the application via a callback. A callback is a function that the application supplies to SslLib that will be called when specific situations occur during the SSL protocol. The callbacks are specific to the particular SslContext or SslLib they are registered with. Examples of callbacks used by the SslLib are an 'information' callback and a 'certificate validation' callback.

SslCipherSuiteInfo Struct

Purpose Encapsulates information about the current cipher suite.

Declared In `SslLib.h`

Prototype

```
typedef struct SslCipherSuiteInfo_st {
   UInt8 cipherSuite[2];
   UInt16 cipher;
   UInt16 digest;
   UInt16 keyExchange;
   UInt16 authentication;
   UInt16 version;
   UInt16 cipherBitLength;
   UInt16 cipherKeyLength;
   UInt16 keyExchangeLength;
   UInt16 authenticationLength;
   UInt16 export;
} SslCipherSuiteInfo
```

Fields

`cipherSuite`
: The two bye cipher suite value for the current cipher suite. See the `CipherSuite` attribute description in the *Palm OS Programmer's Companion*, vol. II, *Communications* for more details.

`cipher`
: A number indicating which cipher is being used for this connection. One of the `sslCsiCipher...` constants listed under "Cipher Suite Info Constants" on page 2206.

`digest`
: The digest value. One of the `sslCsiDigest...` constants listed under "Cipher Suite Info Constants" on page 2206.

`keyExchange`
: The key exchange type which was used. One of the `sslCsiKeyExch...` constants listed under "Cipher Suite Info Constants" on page 2206.

`authentication`
: The authenticaiton type used. One of the `sslCsiAuth...` constants listed under "Cipher Suite Info Constants" on page 2206.

`version`
: The SSL version being used.

`cipherBitLength`
: The number of bits of key material used for encryption key generation. For export ciphers this will be either 40 or 56 bits.

`cipherKeyLength`
: The length of the key for the underlying cipher. For an export RC4 cipher, the `cipherBitLength` would be 40, but the `cipherKeyLength` would be 128. This is because while the SSL protocol would be using 128-bit keys to encrypt and decrypt with RC4, only 40 bits of random data would be used to generate the 128-bit key.

`keyExchangeLength`
: The length in bits of the public key used to establish a shared secret.

`authenticationLength`
: The length of the public key used to ensure the key exchange was not tampered with. For export ciphers, the `keyExchangeLength` is often shorter than the `authenticationLength`.

`export`
: Set when an export cipher is being used.

Comments This structure differs from most others in that the application passes in a structure to be populated from the `SslContext`. Normally the `SslContext` returns a pointer to an internal data structure.

SslContext Struct

Purpose An opaque data structure that represents an SSL connection.

Declared In `SslLib.h`

Prototype

```
typedef struct SslContext_st SslContext
```

Fields None.

SslExtendedItem Struct

Purpose Structure that's used to describe a single certificate-related item.

Declared In `SslLib.h`

Prototype

```
typedef struct SslExtendedItem_st {
   UInt16 type;
   UInt16 field;
   UInt16 dataType;
   UInt16 len;
   UInt32 offset;
} SslExtendedItem
```

Fields

`type`
The extended item type, one of:

`sslExItemTypeX509`
An X.509 certificate.

`sslExItemTypeRSA`
An RSA public key.

`sslExItemTypeRDN`
An X.509 Relative Distinguished Name (RDN). This is the certificate's name. Each certificate contains two names, the *Subject* of the certificate and the *Issuer* of the certificate. Both are encoded as RDNs that contain multiple fields.

`sslExItemTypeX509Ex`
X.509 certificates can contain multiple "extensions." This type is used to specify that the item is a certificate extension.

`field`
Type-specific value.

`dataType`
The encoding for the item's data. For the SSL library, the value is one of the ASN.1 encoding types, as listed in `SslLibAsn1.h`. The encoding is relevant if you're trying to display the data bytes.

`len`
Number of data bytes.

`offset`
: Location of the data itself, as an offset in bytes from the beginning of the `SslExtendedItems` structure that contains this `SslExtendedItem`.

Comments

Every `SslExtendedItem` has a distinct (pre-defined) "extended item type." The location of the item's data is indicated by the structure—the data isn't stored in the structure.

You never create `SslExtendedItem` structures yourself. The objects that relate to a particular certificate are collected into an `SslExtendedItems` structure and returned to you (principally) by your verify callback function.

SslExtendedItems Struct

Purpose

Structure that contains a set of related data items that pertain to an SSL certificate or other cryptographic entity.

Declared In

`SslLib.h`

Prototype

```
typedef struct SslExtendedItems_st {
   UInt32 length;
   UInt32 num;
   SslExtendedItem eitem[1];
} SslExtendedItems
```

Fields

`length`
: Total length of the structure, in bytes.

`num`
: Number of elements in the `eitem` array.

`eitem`
: Individual data items (these are the "related data items" mentioned above). The array contains num elements.

Comments

The `SslExtendedItems` structure is used as part of the `SslVerify` structure, and is used as the type of the `PeerCert` attribute.

The `SslExtendedItems` structure is used by the SSL library to return information to your application; you never create and populate an `SslExtendedItems` structure yourself.

The items in the eitem array needn't all be the same "extended item type" (as defined in SslExtendedItem). For example, the SslExtendedItems structure for a certificate typically contains an sslExItemTypeX509 item, an sslExItemTypeRSA, and an sslExItemType (for the subject name).

SslIoBuf Struct

Purpose Passed to the 'info' functions when I/O is being done.

Declared In SslLib.h

Prototype

```
typedef struct SslIoBuf_st {
   SslContext *ssl;
   UInt8 *ptr;
   UInt32 outNum;
   UInt32 inNum;
   UInt32 max;
   UInt32 err;
   UInt32 flags;
} SslIoBuf
```

Fields

ssl
: The SslContext for which this callback is being called from. It should be the same value that is held in the SslCallback structure's ssl field.

ptr
: The buffer being used to store the bytes.

outNum
: The number of bytes read or written; it is 0 in the 'Before' states.

inNum
: The number of bytes to be read or written.

max
: The maximum read that could be performed. inNum is the number of bytes that the SSL library needs right now, but max bytes could be read if they are available. Often the read operation will read more bytes than are needed, so the outNum field which returns the number of bytes read or written, can, for the read case, be larger than inNum.

`err`
: In the 'After' *`argi`* states, `err` will contain the error code from the I/O operation. If there was no error, it will be 0 (`errNone`).

`flags`
: The flags field is not currently used and is set to zero.

Comments The 'Before' *`argi`* values indicate that the passed `SslIoBuf` indicates the I/O operation about to be performed. The 'After' *`argi`* values indicate that the `SslIoBuf` contains the results of the just-performed I/O operation.

SslLib Struct

Purpose An opaque data structure that represents the SSL Library (similar to "NetLib").

Declared In `SslLib.h`

Prototype
```
typedef struct SslLib_st SslLib
```

Fields None.

SslSession Struct

Purpose Holds all the security information associated with a particular SSL connection.

Declared In `SslLib.h`

Prototype

```
typedef struct SslSession_st {
   UInt32 length;
   UInt16 version;
   unsigned char cipherSuite[2];
   unsigned char compression;
   unsigned char sessionId[33];
   unsigned char masterSecret[48];
   unsigned char time[8];
   unsigned char timeout[4];
   UInt16 certificateOffset;
   UInt16 extraData;
} SslSession
```

Fields

`length`
: The total size of the structure, in bytes.

`version`
: Version number of the SSL protocol that's being used; 0 if the session is using cleartext.

`cipherSuite`
: Cipher suite that's being used. See the `CipherSuite` attribute description in *Palm OS Programmer's Companion*, vol. II, *Communications*.

`compression`
: The name of the compression scheme that's being used.

`sessionId`
: The session ID. The first byte is the length.

`masterSecret`
: Master secret.

`time`
: Host-specific start time.

`timeout`
: Timeout, in seconds.

`certificateOffset`
: Optional Peer certificate; this is the offset from the front of the structure to a `SslExtendedItems` structure. If the offset if 0, it does not exist

`extraData`
: Can be used to store anything, such as the hostname of peer. Application defined.

Comments

A context's `SslSession` is populated when you call `SslOpen()`. You can retrieve a context's `SslSession` structure and modify it directly—particularly if you want to set the fields that are reserved for application use. To retrieve the `SslSession`, call the `SslContextGet_SslSession` macro. Keep in mind that this gives you a pointer to the context's internal `SslSession` attribute. If you want to store the `SslSession`, you must copy the attribute's data.

SslSocket Struct

Purpose

The structure used to hold the Net Library arguments to be passed to any Net Library calls.

Declared In

`SslLib.h`

Prototype

```
typedef struct SslSocket_st {
   NetSocketRef socket;
   UInt16 flags;
   UInt16 addrLen;
   Err err;
   Int32 timeout;
   NetSocketAddrType addr;
} SslSocket
```

Fields

`socket`
: Socket identifier, as created and returned by `NetLibSocketOpen()`. The SSL library doesn't open or connect the socket for you; all socket operations other than reading and writing data must be done through net library calls. Note that you never set the `socket` field directly; use the `SslContextSet_Socket` macro instead.

`flags`
: Options that control a socket's I/O operations. See "I/O Flags" on page 1468 for a description of these options.

`addrLen`
: The length, in bytes, of the `addr` field.

`err`
: The most recent socket error.

`timeout`
: The socket's I/O operation timeout value, in system ticks.

`addr`
: The most recent address that the socket read from or wrote to.

Comments

When setting this structure, the `socket` field will not be copied in. To set the socket to use for network connections, see the Socket attribute. The `socket`, `flags`, `addrLen`, `err`, `timeout`, and `addr` fields all correspond to the arguments passed to `NetLibSend()` and `NetLibReceive()`. Read further on those function for more details. The `SslSocket` passed into the `SslContext` will be copied into the `SslContext`'s `SslSocket` structure and the `SslSocket` pointer returned refers to an internal `SslContext` data structure. When the application calls a function such as `SslReceive()`, the arguments passed to that function will overwrite the internal `SslSocket` values, so a subsequent call to `SslContextGet_IoStruct()` will return the newly updated fields.

SslVerify Struct

Purpose

Structure that's passed to a verify callback function to provide information about a certificate that's being verified. This happens when the verification process hits a snag, giving your app a chance

to the handle the problem. The callback is also invoked when the process has successfully completed.

Declared In `SslLib.h`

Prototype

```
typedef struct SslVerify_st {
   SslExtendedItems *certificate;
   SslExtendedItems *fieldItems;
   UInt32 field;
   SslExtendedItems *ex;
   UInt32 depth;
   UInt32 state;
} SslVerify
```

Fields

`certificate`
: Points to the certificate that's being verified.

`fieldItems`
: An array of extended items. The `SslExtendedItems` that's located at `fieldItems[field]` contains the set of items that's causing the problem.

`field`
: An index into `fieldItems`.

`ex`
: If this is an extension error, the `ex` field contains the data element of the X509 extension that failed.

`depth`
: This is the level of the certificate being processed, where 0 is the server's certificate, and higher numbers are certificates along the "certificate chain" that leads to a trusted root certificate.

`state`
: The verification state.

Comments The list of `SslExtendedItem` elements that the `SslExtendedItems` object located at `fieldItems[ field ]` contains depends on the verification error:

Flavor	Contents
`sslErrVerifyBadSignature`	The server's certificate, which contains the public key entries.
`sslErrVerifyNoTrustedRoot`	`NULL`
`sslErrVerifyNotAfter`	`sslExItemTypeX509`, `asn1FldX509NotAfter`
`sslErrVerifyNotBefore`	`sslExItemTypeX509`, `asn1FldX509NotBefore`.
`sslErrVerifyConstraintViolation`	`asn1ExItemTypeX509Ex` (but see below)
`sslErrVerifyUnknownCriticalExtension`	`asn1ExItemTypeX509Ex` (but see below)
`sslErrOk`	`NULL`

The `asn1ExItemTypeX509Ex` item needs to be decoded in order to be used. The item contains the "object identifier" part of a certificate extension. The item in the next field (i.e. `fieldItems[ field+1 ]`) depends on the item's `dataType` field: If it's `asn1Boolean`, than the "+1" item contains the extension's optional `Boolean` "this is a critical extension" field, otherwise it contains the extension data itself. If "+1" contains the `Boolean`, than "+2" is the data.

The SSL library attempts to interpret only critical extensions, so the critical field should always be present. If a critical extension isn't understood, the certificate should be rejected.

SSL Library Constants

SSL Open Mode Flags

Purpose Flags used to specify how the connection should be started when calling `SslOpen()`.

Declared In `SslLib.h`

Constants

`#define sslOpenBufferedReuse 0x0040`
This is the same as `SslContextSet_BufferedReuse(ssl,1);`

`#define sslOpenDelayHandshake 0x0080`
Do not perform the handshake now

`#define sslOpenModeClear 0x0001`
Turn off the SSL protocol.

`#define sslOpenModeSsl 0x0002`
Turn on the SSL protocol.

`#define sslOpenNewConnection 0x0004`
Perform a new SSL handhake, clearing any previous `SslSession` value. This is the same as `SslContextSet_SslSession(ssl,NULL)`.

`#define sslOpenNoAutoFlush 0x0008`
This is the same as `SslContextSet_AutoFlush(ssl,0);`

`#define sslOpenUseDefaultTimeout 0x0020`
Use the `SslContext` timeout value instead of timeout parameter.

SSL Close Mode Flags

Purpose Flags that allow you to specify how to perform the SSL Protocol shutdown when calling `SslClose()`.

Declared In `SslLib.h`

Constants

`#define sslCloseDontSendShutdown 0x0001`
Do not send the SSL shutdown message to the server.

`SslContextSet_DontSendShutdown(ssl,1);`

`#define sslCloseDontWaitForShutdown 0x0002`
Don't wait for the server to send a shutdown message.

`SslContextSet_DontWaitForShutdown(ssl,1);`

`#define sslCloseUseDefaultTimeout 0x0020`
Use the timeout value set against the SslContext, not the *`timeout`* parameter.

Mode Attribute Values

Purpose Values that control the SslContext's operating mode.

Declared In `SslLib.h`

Constants `#define sslModeClear 0x0000`
Causes the SSL protocol to be bypassed.

`#define sslModeFlush 0x8000`
Causes any data in the internal data buffers to be cleared.

`#define sslModeSsl 0x0002`
A subset value of `sslModeSslClient`

`#define sslModeSslClient 0x000A`
SSL is turned on, with this endpoint a client.

Comments `SslModeSsl` is a subset value of `sslModeSslClient`. In a future release of SslLib, the server side of the SSL protocol may be supported in which case `sslModeSslServer` would be added. The application can use code like the following to determine if the SSL protocol is being used:

```
If (SslContextGet_Mode(ssl) & sslModeSsl)
   /* SSL protocol enabled */
else
   /* Using cleartext */
```

A comparison with `sslModeSslClient` could be used to determine if the client or server side of the protocol is being used for that particular SslContext.

The `sslModeFlush` flag is special. When used in `SslContextSet_Mode()`, it causes any data in the internal data buffers to be cleared. This is normally required when reusing a SslContext for a new connection. If an application is using a

SslContext for cleartext, and then wants to enable SSL on the same connection, this flag should not be used.

Protocol Versions

Purpose The version of the SSL protocol to use.

Declared In `SslLib.h`

Constants

`#define sslVersionSSLv3 0x0300`
Version SSLv3 of the SSL protocol.

Comments SslLib implements only SSLv3 at this point in time, so modification of this value is not a good idea.

Compatibility Flags

Purpose `Compat` attribute flags that turn on compatiblity with incorrect SSL protocol implementations.

Declared In `SslLib.h`

Constants

`#define sslCompat1RecordPerMessage 0x0004`
Some servers do not like to receive SSL protocol messages separated into multiple SSL records. This option stops the write buffers being set to a size less than 1024 bytes which ensures this problem will not occur.

`#define sslCompatAll 0xffff`
This value enables all the bug compatibility flags.

`#define sslCompatBigRecords 0x0008`
Some old Microsoft servers would send data to the SSL Client in records larger than 16k bytes in size. This is not legal in the SSLv3 protocol. This flag makes SslLib tolerate these large records.

`#define sslCompatNetscapeCaDnBug 0x0002`
Enables support for some old versions of Netscape servers which encoded certificate requests Distinguished names wrongly. This is not currently a problem for SslLib since it does not support Client certificates.

`#define sslCompatReuseCipherBug 0x0001`
Enables support for servers that change cipher suites on session-reuse. They should not be doing this.

Comments These bugs will not normally be encountered while using the SSL protocol, but if desired, it is worth enabling the compatibility in case old buggy servers are being accessed.

SSL Callback Commands

Purpose General 'commands' that all callbacks should expect to receive. There are normally related to creation and destruction of the structure that holds the callback. These commands are used in conjunction with `SslCallbackFunc()`.

Declared In `SslLib.h`

Constants

`#define sslCmdFree 0x0002`
Called when the callback is 'destroyed', normally due to its parent SslLib or SslContext being destroyed.

`#define sslCmdGet 0x0004`
Called to return a value from the callback to SslLib. The *argi* value for this callback is callback-specific.

`#define sslCmdInfo 0x0012`
An Info callback. See "The Info Callback" on page 2242 for more information.

`#define sslCmdNew 0x0001`
Called when the callback is 'copied' into an SslLib or SslContext.

`#define sslCmdRead 0x0010`

`#define sslCmdReset 0x0003`
Called when the SslContext has been reset, which occurs when a new connection is being started. It is called instead of `sslCmdNew`. `sslCmdNew` will be called only once; `sslCmdReset` will be called subsequently to 'reset' the state associated with the callback.

`#define sslCmdSet 0x0005`
Called to pass a value from SslLib to the callback. The *argi* value would specify the *argv* parameter. The *argi* parameter would be callback-specific.

`#define sslCmdVerify 0x0013`
A Verify callback. See "The Verify Callback" on page 2243 for more information.

`#define sslCmdWrite 0x0011`

Cipher Suite Info Constants

Purpose Constants used with various fields of the `SslCipherSuiteInfo` structure.

Declared In `SslLib.h`

Constants

`#define sslCsiAuthNULL 0x00`
(`authentication` field) No key exchange.

`#define sslCsiAuthRsa 0x01`
(`authentication` field) RSA.

`#define sslCsiCipherNull 0x00`
(`cipher` field) No cipher is being used for this connection.

`#define sslCsiCipherRc4 0x01`
(`cipher` field) The RSA RC4 cipher is being used for this connection.

`#define sslCsiDigestMd2 0x03`
(`digest` field) Rivest Message Digest 2.

`#define sslCsiDigestMd5 0x01`
(`digest` field) Rivest Message Digest 5.

`#define sslCsiDigestNull 0x00`
(`digest` field) No message digest.

`#define sslCsiDigestSha1 0x02`
(`digest` field) Secure Hash Algorithm-1.

`#define sslCsiKeyExchNull 0x00`
(`keyExchange` field) No key exchange is being used.

`#define sslCsiKeyExchRsa 0x01`
(`keyExchange` field) RSA key exchange.

Cipher Suites

Purpose SSL cipher suites that the SSL protocol can attempt to use.

Declared In `SslLib.h`

Constants

```
#define sslCs_RSA_RC4_128_MD5 0x00,0x04
#define sslCs_RSA_RC4_128_SHA1 0x00,0x05
#define sslCs_RSA_RC4_40_MD5 0x00,0x03
#define sslCs_RSA_RC4_56_SHA1 0x00,0x64
```

Comments Set the cipher suites using either `SslLibSet_CipherSuites()` or `SslContextSet_CipherSuites()`.

Info Callbacks

Purpose The `SslCallbackFunc()` callback is called when various situations occur during the usage of a SslContext. It is primarily intended for debugging and feedback purposes. If the callback returns a non-zero value, this error will be returned back out the SslLib API. The callback will be called with a command argument of `sslCmdInfo`. The constants listed in this section represent the possible *argi* values.

Declared In `SslLib.h`

Constants

`#define sslArgInfoAlert 0x0002`

Notification of an Alert in the SSL protocol. The `sslArgInfoAlert` notification is called with a `NULL` value for the *argv* parameter. The application can get the `LastAlert` attribute from the `SslContext` to determine which alert was received.

`#define sslArgInfoCert 0x0003`

Notification of peer certificate. The `sslArgInfoCert` call is made after the server's certificate chain has been verified. The *argv* parameter is a `SslExtendedItems` pointer, which points to the remote server's certificate.

`#define sslArgInfoHandshake 0x0001`

Notification of a state change in the SSL protocol. The `sslArgInfoHandshake` will be called upon each handshake state change. The *argv* parameter will be `NULL`,

but the `HsState` attribute can be interrogated to read the current state.

`#define sslArgInfoReadAfter (sslCmdRead | 0x8000)`
Notification after a Receive/Read Net Library call. See the Comments section, below, for more information.

`#define sslArgInfoReadBefore sslCmdRead`
Notification before a Receive/Read Net Library call. See the Comments section, below, for more information.

`#define sslArgInfoWriteAfter (sslCmdWrite | 0x8000)`
Notification after a Send/Write Net Library call. See the Comments section, below, for more information.

`#define sslArgInfoWriteBefore sslCmdWrite`
Notification before a Send/Write Net Library call. See the Comments section, below, for more information.

Comments

The `sslArgInfo[Read | Write] [Before | After]` callback is called twice for each network I/O operation. The first call is made before the call to the underlying Net Library Send/Receive. The second is made after the call has completed. If the callback returns a non-zero value, this value will be returned by origional SslLib call the application made.

The *argv* parameter is a `SslIoBuf` structure. This structure's `ssl` field is the SslContext that the I/O operation is being performed by. `ptr` points to the space used, or to be used, in the operation. `outNum` is the number of bytes processed. It is only set in the After calls. `inNum` is the number of bytes to be read or written in the call. `max` is the maximum number of bytes that could be read. It can be larger than `inNum`. `err` is the error value, if any. This value is only set in the After calls. `flags` is currently not used and is set to 0.

InfoInterest Values

Purpose Values used to specify the events for which of the Info Callbacks will be called. The `InfoInterest` value is the logical ORing of these values.

Declared In `SslLib.h`

Constants

`#define sslFlgInfoAlert 0x0001`
The `sslArgInfoAlert` callback.

`#define sslFlgInfoCert 0x0008`
The `sslArgInfoCert` callback.

`#define sslFlgInfoHandshake 0x0002`
The `sslArgInfoHandshake` callback.

`#define sslFlgInfoIo 0x0004`
The `sslArgInfoReadAfter`, `sslArgInfoReadBefore`, `sslArgInfoWriteAfter`, and `sslArgInfoWriteBefore` callbacks.

LastApi Attribute Values

Purpose The last SslLib API call made. This attribute can be useful in event-driven programs.

Declared In `SslLib.h`

`#define sslLastApiFlush 0x04`
The previous function was `SslFlush()`.

`#define sslLastApiNone 0x00`
No SSL library function call in this context since the last SSL reset.

`#define sslLastApiOpen 0x01`
The previous function was `SslOpen()`.

`#define sslLastApiRead 0x02`
Set if `SslRead()`, `SslPeek()` or `SslReceive()` was last called.

`#define sslLastApiShutdown 0x05`
The previous function was `SslClose()`.

`#define sslLastApiWrite 0x03`
Set if `SslWrite()` or `SslSend()` was last called.

LastIO Attribute Values

Purpose The last network operation performed.

Declared In `SslLib.h`

`#define sslLastIoNone 0x00`
No I/O operations have been performed since the context was last reset.

`#define sslLastIoRead 0x01`
A read operation.

`#define sslLastIoWrite 0x02`
A write operation.

Comments Since most of the SslLib API I/O functions can cause an SSL handshake to be performed, it is often not possible to know if the reason that a `SslSend()` returned `netErrWouldBlock` is because the Send operation failed or a Receive operation failed (because a SSL Handshake was being performed). This attribute allows the application to determine which I/O operation was being called if a network error is returned. If the application is using `NetLibSelect()`, this attribute is very important. Because this attribute returns the last network operation performed, `sslLastIoNone` will only be returned if the SslContext has not performed any I/O operations since its last reset.

SSL Protocol States

Purpose These constants indicate the state that the SSL protocol is currently in. See the SSL protocol specification for clarification on what the values mean.

Declared In `SslLib.h`

Constants

```
#define sslHsStateCert 7
#define sslHsStateCertB 8
#define sslHsStateCertReq 13
#define sslHsStateCertReqB 14
#define sslHsStateCkEx 17
#define sslHsStateCleanup 25
```

```
#define sslHsStateClientCert 16
#define sslHsStateClientHello 2
#define sslHsStateClosed 28
#define sslHsStateDone 26
#define sslHsStateFinished 19
#define sslHsStateFlush 4
#define sslHsStateGenerateKeys 21
#define sslHsStateHelloRequest 29
#define sslHsStateNone 0
#define sslHsStateReadCcs 20
#define sslHsStateReadFinished 22
#define sslHsStateReadFinishedB 23
#define sslHsStateReadFinishedC 24
#define sslHsStateServerDone 15
#define sslHsStateServerHello 3
#define sslHsStateShutdown 27
#define sslHsStateSkEx 9
#define sslHsStateSkExAnonDh 12
#define sslHsStateSkExDh 11
#define sslHsStateSkExRsa 10
#define sslHsStateStart 1
#define sslHsStateWrite 6
#define sslHsStateWriteCcs 18
#define sslHsStateWriteClose 30
#define sslHsStateWriteFlush 5
```

SSL Server Alerts

Purpose Alert values received from the server.

Declared In `SslLib.h`

Constants

```
#define sslAlertAccessDenied (0x0200+49)

#define sslAlertBadCertificate (0x0100+42)

#define sslAlertBadRecordMac (0x0200+20)

#define sslAlertCertificateExpired (0x0100+45)

#define sslAlertCertificateRevoked (0x0100+44)

#define sslAlertCertificateUnknown (0x0100+46)

#define sslAlertCloseNotify (0x0100+ 0)

#define sslAlertDecodeError (0x0200+50)

#define sslAlertDecompressionFailure (0x0200+30)

#define sslAlertDecryptError (0x0200+51)

#define sslAlertDecryptionFailed (0x0200+21)

#define sslAlertExportRestricion (0x0200+60)

#define sslAlertHandshakeFailure (0x0200+40)

#define sslAlertIllegalParameter (0x0200+47)

#define sslAlertInsufficientSecurity (0x0200+71)
```

```
#define sslAlertInternalError (0x0200+80)

#define sslAlertNoCertificate (0x0100+41)

#define sslAlertNoRenegotiation (0x0100+100)

#define sslAlertProtocolVersion (0x0200+70)

#define sslAlertRecordOverflow (0x0200+22)

#define sslAlertUnexpectedMessage (0x0200+10)

#define sslAlertUnknownCa (0x0200+48)

#define sslAlertUnsupportedCertificate (0x0100+43)

#define sslAlertUserCancled (0x0100+90)
```

Comments The alert values are received from the server and are of two types, fatal and non-fatal.

The non-fatal alerts have a value of the form `0x01`*XX*, while fatal Alerts have the form `0x02`*XX*. SslLib will fail on fatal alerts and continue on non-fatal alerts.

Verification States

Purpose These constants represent the possible values of the `SslVerify` structure's `state` field. They indicate which section of the certificate verification failed.

Declared In `SslLib.h`

Constants

```
#define sslVerifyDone 6

#define sslVerifyExtensions 5
```

```
#define sslVerifyFindParent 1

#define sslVerifyNotAfterFindParent 4

#define sslVerifyNotBefore 3

#define sslVerifySignature 2
```

SSL Library Errors

Purpose Error codes returned by the various SSL Library functions.

Declared In `SslLib.h`

Constants

`#define sslErrBadArgument (sslErrorClass+17)`
An invalid argument was provided to the function.

`#define sslErrBadDecode (sslErrorClass+9)`
An error occurred while decoding values during certificate verification.

`#define sslErrBadLength (sslErrorClass+13)`
A length argument was invalid.

`#define sslErrBadOption (sslErrorClass+18)`
An invalid argument was provided to the function.

`#define sslErrBadPeerFinished (sslErrorClass+46)`
The final check of the SSL handshake failed. This indicates that there was a problem establishing a shared secret value. It could be caused by the server using a certificate that does not match its private key.

`#define sslErrBadSignature (sslErrorClass+47)`
An invalid signature was found on a ephemeral Cipher Suite message.

`#define sslErrBufferTooSmall (sslErrorClass+11)`
A supplied buffer was not large enough for the output data.

`#define sslErrCbAbort (sslErrorClass+4)`
This error code would be returned by an applications callback function to indicate a desire to exit. This error may

not be fatal, depending on the callback that generated the error.

`#define sslErrCert (sslErrorClass+39)`
A generic error occurred inside the SslLib certificate library.

`#define sslErrCertDecodeError (sslErrorClass+51)`
The Servers certificate could not be decoded.

`#define sslErrCsp (sslErrorClass+38)`
A generic error occurred inside the SslLib cryptographic library.

`#define sslErrDivByZero (sslErrorClass+7)`
Something went wrong in the Maths library. These error will normally only be generated by certificates which have invalid public keys.

`#define sslErrEof (sslErrorClass+2)`
Error returned by SslLib functions when either the SSL protocol has been closed or the underlying socket has been closed. This error indicates that the current SslContext is unable to read or write any more data bytes.

```
#define sslErrExtraHandshakeData
  (sslErrorClass+43)
```
Extra data was found in the SSL handshake messages that should not have been there.

`#define sslErrFailed (sslErrorClass+1)`
A generic error.

`#define sslErrFatalAlert (sslErrorClass+45)`
A fatal alert was received by the SSL protocol.

`#define sslErrHandshakeEncoding (sslErrorClass+40)`
An error occurred during decoding of SSL handshake messages.

`#define sslErrHandshakeProtocol (sslErrorClass+42)`
An error occurred while processing the decoded SSL handshake messages.

`#define sslErrInitNotCalled (sslErrorClass+10)`
An internal SslLib error.

`#define sslErrInternalError (sslErrorClass+21)`
An internal SslLib error.

`#define sslErrIo (sslErrorClass+5)`
This error code is returned when an underlying Net Library function call returned an error that is not fatal. A timeout, or other such non-fatal network errors will be reclassified as this error type. A function that returns this error type can be re-called once the error condition has disappeared.

`#define sslErrMissingProvider (sslErrorClass+41)`
An internal SslLib error.

`#define sslErrNoDmem (sslErrorClass+14)`
An internal SslLib error.

`#define sslErrNoMethodSet (sslErrorClass+15)`
An internal SslLib error.

`#define sslErrNoModInverse (sslErrorClass+8)`
Something went wrong in the Maths library. These error will normally only be generated by certificates which have invalid public keys.

`#define sslErrNoRandom (sslErrorClass+16)`
A problem with the random number source.

`#define sslErrNotFound (sslErrorClass+6)`
Returned on an internal SslLib search that did not find a valid entry. Consider this an internal SslLib error.

`#define sslErrNotImplemented (sslErrorClass+19)`
An internal SslLib error.

`#define sslErrNullArg (sslErrorClass+12)`
An passed argument was `NULL` that should not have been `NULL`.

`#define sslErrOk (sslErrorClass+0)`
Not an error.

`#define sslErrOutOfMemory (sslErrorClass+3)`
Returned if a dynamic memory allocation failed. This is normally considered a very bad error.

`#define sslErrReadAppData (sslErrorClass+50)`
Application data was read by the SSL protocol when it was expecting handshake messages.

`#define sslErrReallocStaticData (sslErrorClass+20)`
An internal SslLib error

```
#define sslErrRecordError (sslErrorClass+37)
```
An invalid record was received in the SslContext.

```
#define sslErrUnexpectedRecord (sslErrorClass+49)
```
A record of the wrong was received inside the SSL protocol .

```
#define sslErrUnsupportedCertType
  (sslErrorClass+52)
```
The Servers certificate contains a public key we cannot decode.

```
#define sslErrUnsupportedSignatureType
  (sslErrorClass+53)
```
We have been send a certificate with a signature type we do not recognize.

```
#define sslErrVerifyBadSignature
  (sslErrorClass+128)

#define sslErrVerifyCallback (sslErrorClass+134)

#define sslErrVerifyConstraintViolation
  (sslErrorClass+132)

#define sslErrVerifyNotAfter (sslErrorClass+130)

#define sslErrVerifyNotBefore (sslErrorClass+131)

#define sslErrVerifyNoTrustedRoot
  (sslErrorClass+129)

#define sslErrVerifyUnknownCriticalExtension
  (sslErrorClass+133)

#define sslErrWrongMessage (sslErrorClass+44)
```
An invalid SSL message was received.

SSL Library Traps

Purpose These constants are used by the trap mechanism to call the various functions in the SSL library.

Declared In `SslLib.h`

Constants

```
#define kSslClose (sysLibTrapCustom+13)
#define kSslConsume (sysLibTrapCustom+19)
#define kSslContextCreate (sysLibTrapCustom+6)
#define kSslContextDestroy (sysLibTrapCustom+7)
#define kSslContextGetLong (sysLibTrapCustom+8)
#define kSslContextGetPtr (sysLibTrapCustom+10)
#define kSslContextSetLong (sysLibTrapCustom+9)
#define kSslContextSetPtr (sysLibTrapCustom+11)
#define kSslFlush (sysLibTrapCustom+20)
#define kSslLibClose sysLibTrapClose
#define kSslLibCreate (sysLibTrapCustom)
#define kSslLibDestroy (sysLibTrapCustom+1)
#define kSslLibGetLong (sysLibTrapCustom+3)
#define kSslLibGetPtr (sysLibTrapCustom+5)
#define kSslLibName sysLibTrapName
#define kSslLibOpen sysLibTrapOpen
#define kSslLibSetLong (sysLibTrapCustom+2)
#define kSslLibSetPtr (sysLibTrapCustom+4)
#define kSslLibSleep sysLibTrapSleep
#define kSslLibWake sysLibTrapWake
#define kSslOpen (sysLibTrapCustom+12)
#define kSslPeek (sysLibTrapCustom+18)
#define kSslRead (sysLibTrapCustom+16)
#define kSslReceive (sysLibTrapCustom+15)
```

```
#define kSslSend (sysLibTrapCustom+14)
#define kSslWrite (sysLibTrapCustom+17)
```

Miscellaneous SSL Library Constants

Purpose These constants are also defined in `SslLib.h`.

Declared In `SslLib.h`

Constants

`#define kSslDBName "SslLib"`
The SSL library's name. Use this when determining if the SSL library is loaded with `SysLibFind()`.

`#define kSslLibCreator 'ssl0'`
The SSL library's creator ID. Use this when loading the SSL library with `SysLibLoad()`.

`#define kSslLibType sysFileTLibrary`
The SSL library's type. Use this when loading the SSL library with `SysLibLoad()`.

SSL Library Functions and Macros

SslClose Function

Purpose Performs the shutdown part of the SSL protocol.

Declared In `SslLib.h`

Prototype

```
Err SslClose (UInt16 libRef, SslContext *ctx,
   UInt16 mode, UInt32 timeout)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ctx*
The context to query.

→ *mode*
Flags that specify how to perform the SSL Protocol shutdown. A combination of the values listed under "SSL Close Mode Flags" on page 2202.

→ *timeout*
Timeout, in system ticks, to use for final message exchange

Returns Returns `errNone` if the operation completed successfully. Otherwise, this function returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments This usually involves message exchanges. This function can be repeatedly called after a timeout until either a network error is reported or the final SSL shutdown message exchange has been completed. The mode values can be logically ORed together.

Example The following code excerpt show how you might use this function:

```
Err err;
SslContext *ssl;

err = SslOpen(theLibRef, ssl,0,20*SysTicksPerSecond());
/* Perform SSL IO */
/* Shutdown the protocol but don't linger waiting for a
 * response from the server */
err = SslClose(theLibRef, ssl,sslCloseDontWaitForshutdown,
   20*SysTicksPerSecond());
```

See Also `SslOpen()`, `SslContextSetLong()`, `SslContextSetPtr()`

SslConsume Function

Purpose Removes up to a specified number of bytes from the buffered read bytes in the passed `SslContext`.

Declared In `SslLib.h`

Prototype
```
void SslConsume (UInt16 libRef, SslContext *ctx,
    Int32 number)
```

Parameters → *libRef*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ctx*
The SslContext to operate on.

→ *number*
The number of bytes to remove from the internal buffer.

Returns Nothing.

Comments This function is normally used in conjunction with `SslPeek()`.

Example The following code excerpt shows how this function might be used.

```
Err err;
void *data;
Int32 *dataLen;

err=SslPeek(theLibRef, ssl,&data,*dataLen,16*1024);
/* Process the dataLen bytes located at data */
SslConsume(theLibRef, ssl,dataLen);
```

See Also `SslPeek()`, `SslRead()`, `SslReceive()`

SslContextCreate Function

Purpose Creates a new SSL Context.

Declared In `SslLib.h`

Prototype

```
Err SslContextCreate (UInt16 libRef, SslLib *lib,
    SslContext **ctx)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SSL library structure.

← *ctx*
Where to deposit the SslContext pointer.

Returns Returns `errNone` if the operation completed successfully. Otherwise, this function returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments A SslContext is the data structure used to encapsulates all aspects of a SSL connection. This routine will deposit a pointer to the newly created structure at the address given by the *ctx* argument. Various default values will be inherited from the passed SslLib.

See Also SslOpen(), SslContextDestroy(), SslContextSetLong(), SslContextSetPtr()

SslContextDestroy Function

Purpose Destroys the SSL Context.

Declared In SslLib.h

Prototype

```
void SslContextDestroy (UInt16 libRef,
    SslContext *ctx)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either SysLibFind() or SysLibLoad(), depending on whether or not the SSL library is already loaded.

→ *ctx*
The SslContext to destroy.

Returns Nothing.

Comments This routine will free the memory associated with the passed SslContext. This function will not close the network connection or shutdown the SSL protocol. See SslClose() for information on shutting down the SSL Protocol.

See Also SslContextCreate(), SslClose()

SslContextGetLong Function

Purpose Retrieve an integer attribute value from the passed SslContext structure.

Declared In `SslLib.h`

Prototype

```
Int32 SslContextGetLong (UInt16 libRef,
   SslContext *lib, SslAttribute attr)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either SysLibFind() or SysLibLoad(), depending on whether or not the SSL library is already loaded.

→ *lib*
The SslContext from which the value is to be retrieved.

→ *attr*
Attribute to retrieve.

Returns The value of the attribute is returned. If a non-existent attribute was requested, -1 is returned. This could give incorrect values so an application should make sure to call this routine with the correct arguments.

Comments This function is not normally used directly, but via pre-defined macros.

Example The following example shows the use of one of the macros that make use of this function:

```
/* Is the SslContext configured to do ssl? */
if (!(SslContextGet_Mode(theLibRef, lib) & sslModeSsl))
   return(WE_ARE_NOT_USING_SSL);
```

See Also SslContextSetLong(), SslContextGetLong()

SslContextGetPtr Function

Purpose Retrieve a pointer to an attribute value from the passed SslContext structure.

Declared In `SslLib.h`

Prototype

```
Err SslContextGetPtr (UInt16 libRef,
    SslContext *lib, SslAttribute attr,
    void **value)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslContext to retrieve the attribute from.

→ *attr*
The attribute to retrieve

← *value*
A pointer to an attribute specific pointer

Returns Returns `errNone` if the operation completed successfully. Otherwise, this function returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments This function is not normally used directly, but via pre-defined macros. The type of the pointer returned is specific to the attribute being requested.

Example The following example shows the use of one of the macros that makes use of this function:

```
SslSession *session;
Err err;

err = SslContextGet_SslSession(theLibRef, ssl,&session);
```

See Also `SslContextSetPtr()`, `SslLibGetPtr()`

SslContextSetLong Function

Purpose Modify one of the numeric attributes of a SslContext structure.

Declared In `SslLib.h`

Prototype

```
Err SslContextSetLong (UInt16 libRef,
    SslContext *lib, SslAttribute attr,
    long value)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslContext on which to operate.

→ *attr*
The attribute to modify.

→ *value*
The new value.

Returns Returns `errNone` if the operation completed successfully. Otherwise, this function returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments This function is not normally used directly, rather it is used via pre-defined macros. The *attr* parameter specifies the SslContext attribute that will be set to the value passed in *value*.

Example The following code excerpt shows how this function is used.

```
SslContext *ssl;
Err err;

err = SslContextCreate(theLibRef, lib,&ssl);
/*modify output buffer size */
err = SslContextSet_WbufSize(theLibRef, lib, 8*1024);
```

See Also `SslContextSetPtr()`, `SslContextGetPtr()`, `SslContextGetLong()`

SslContextSetPtr Function

Purpose Update one of the non-integer attributes of a SslContext.

Declared In `SslLib.h`

Prototype

```
Err SslContextSetPtr (UInt16 libRef,
    SslContext *lib, SslAttribute attr,
    void *value)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslContext to modify.

→ *attr*
The attribute to update.

→ *value*
The value to update, specific to the SslAttribute.

Returns Returns `errNone` if the operation completed successfully. Otherwise, this function returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments The *attr* value defines the type of the *value* parameter. This function is not normally used directly, rather it is used via pre-defined macros.

Example The following code excerpt shows how this function is used.

```
SslContext *ssl;
SslCallback cb;
Err err;

err = SslContextCreate(theLibRef, lib,&ssl);

/* Configure to have 'debugCallbackFunction' called for each
 * SSL protocol handshake state change */
cb.callback = debugCallbackFunction;
cb.data = NULL;
```

```
err = SslContextSet_InfoInterest(theLibRef, lib,
   sslInfoHandshake);
err = SslContextSet_InfoCallback(theLibRef, lib,(void *)&cb);
```

See Also SslContextGetLong(), SslContextSetLong(), SslContextGetPtr()

SslFlush Function

Purpose Cause an immediate write of any data buffered in the SslContext to the network.

Declared In `SslLib.h`

Prototype

```
Err SslFlush (UInt16 libRef, SslContext *ctx,
    Int32 *outstanding)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either SysLibFind() or SysLibLoad(), depending on whether or not the SSL library is already loaded.

→ *ctx*
The SslContext to operate on.

← *outstanding*
The number of byte still unflushed after this call.

Returns Returns `errNone` if the operation completed successfully. Otherwise, this function returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments A SslContext can be set into 'noAutoFlush' mode. This means that the SslSend and SslWrite operations will not cause an immediate write to the network. If this mode is enabled, then explicit SslFlush calls need to be made to ensure that the data buffered in the SslContext is sent to the network. The main use of noAutoFlush is to allow multiple SslWrite/SslSend commands to have their output buffered in the SslContexts output buffer. This improves the SSL Protocols efficiency and is generally a good policy if lots of small write operations are being performed by the application. The number of bytes that can be written to the SslContext write buffer is a few tens of bytes less than the output buffer size. This means that if the application is writing less than this number of bytes, no

network errors can occur until the SslFlush call is made. The outstanding parameter will be updated to contain the number of buffered bytes that are still buffered in the SslContext. If this value is non-zero, the next SslWrite, SslSend, SslFlush operaton will attempt to write those bytes to the network.

Example The following code excerpt shows how this function can be used:

```
Err err;

SslContextSet_AutoFlush(theLibRef, ssl,0);
SslWrite(theLibRef, ssl,"GET ",4, &err);
SslWrite(theLibRef, ssl,url,StrLen(url), &err);
SslWrite(theLibRef, ssl," HTTP/1.0\r\n\r\n",13, &err);

err=SslFlush(theLibRef, ssl,NULL);
```

See Also SslWrite(), SslSend(), SslContextSetLong()

SslLibClose Function

Purpose SSL Library's shared library close function.

Declared In `SslLib.h`

Prototype `Err SslLibClose (UInt16 libRef)`

Parameters → *libRef*

SSL library reference number, obtained from a call to either SysLibFind() or SysLibLoad(), depending on whether or not the SSL library is already loaded.

Returns Returns `errNone` if the SSL library was successfully closed.

SslLibCreate Function

Purpose Creates the SSL library context.

Declared In `SslLib.h`

Prototype
```
Err SslLibCreate (UInt16 libRef, SslLib **lib)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

← *lib*
Where to deposit the SslLib pointer

Returns Returns `errNone` if the operation completed successfully. Otherwise, this function returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments This routine will deposit the newly-created SslLib at the address given by the *lib* argument. This routine is generally the first call made when performing SSL functionality. Various default configuration values can be set against this structure. These values will be inherited by SslContext structures created against the SslLib.

See Also `SslLibDestroy()`, `SslContextCreate()`, `SslLibSetLong()`, `SslLibSetPtr()`

SslLibDestroy Function

Purpose Destroys the context represented by *lib*.

Declared In `SslLib.h`

Prototype
```
void SslLibDestroy (UInt16 libRef, SslLib *lib)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
SslLib structure to be destroyed.

Returns Nothing.

See Also `SslLibCreate()`

SslLibGetLong Function

Purpose Retrieve an integer attribute value from the passed SslLib structure.

Declared In `SslLib.h`

Prototype

```
Int32 SslLibGetLong (UInt16 libRef, SslLib *lib,
    SslAttribute attr)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either SysLibFind() or SysLibLoad(), depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib from which the value is to be retrieved.

→ *attr*
Attribute to retrieve.

Returns The value of the attribute is returned. If a non-existent attribute was requested, -1 is returned.

Comments This function is not normally used directly, but via pre-defined macros.

See Also SslContextGetPtr(), SslLibSetLong(), SslContextGetLong()

SslLibGetPtr Function

Purpose Retrieve an pointer attribute value from the passed SslLib structure.

Declared In `SslLib.h`

Prototype

```
Err SslLibGetPtr (UInt16 libRef, SslLib *lib,
    SslAttribute attr, void **value)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either SysLibFind() or SysLibLoad(), depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib to retrieve the attribute from.

→ *attr*
The attribute to retrieve.

← *value*
A pointer to a attribute specific pointer

Returns Returns `errNone` if the operation completed successfully. Otherwise, this function returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments This function is not normally used directly, but via pre-defined macros. The type of the pointer returned is specific to the attribute being requested.

Example The following code excerpt shows how this function is used.

```
SslCallback *cb;
Err err;

err = SslLibGet_InfoCallback(theLibRef, lib,&cb);
```

See Also `SslLibSetPtr()`, `SslContextGetPtr()`

SslLibName Function

Purpose

Declared In `SslLib.h`

Prototype `Err SslLibName (UInt16 libRef)`

Parameters → *libRef*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

Returns

SslLibOpen Function

Purpose SSL Library's shared library open function.

Declared In `SslLib.h`

Prototype

```
Err SslLibOpen (UInt16 libRef)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either SysLibFind() or SysLibLoad(), depending on whether or not the SSL library is already loaded.

Returns Returns `errNone` if the SSL library was successfully opened.

SslLibSetLong Function

Purpose Modify one of the numeric attributes of a SslLib structure.

Declared In `SslLib.h`

Prototype

```
Err SslLibSetLong (UInt16 libRef, SslLib *lib,
    SslAttribute attr, Int32 value)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either SysLibFind() or SysLibLoad(), depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib on which to operate.

→ *attr*
The attribute to modify.

→ *value*
The new value.

Returns Returns `errNone` if the operation completed successfully. Otherwise, this function returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments This function is not normally used directly, rather it is used via pre-defined macros. The *attr* parameter specifies the SslLib attribute that will be set to the value passed in *value*.

Example The following code excerpt shows how this function is used.

```
SslLib *lib;
Err err;

err = SslLibCreate(theLibRef, &lib);
err = SslLibSet_AutoFlush(theLibRef, lib,0); /* Turn off */
```

See Also SslLibCreate(), SslLibSetPtr(), SslLibGetPtr(), SslLibGetLong()

SslLibSetPtr Function

Purpose Update one of the non-integer attributes of a SslLib.

Declared In SslLib.h

Prototype

```
Err SslLibSetPtr (UInt16 libRef, SslLib *lib,
    SslAttribute attr, void *value)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either SysLibFind() or SysLibLoad(), depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib to operate on.

→ *attr*
The attribute to update.

→ *value*
The value to update, specific to the SslAttribute.

Returns Returns errNone if the operation completed successfully. Otherwise, this function returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments The *attr* value defines the type of the *value* parameter. This function is not normally used directly, rather it is used via pre-defined macros.

Example The following code excerpt shows how this function is used.

```
SslLib *lib;
SslCallback cb;
Err err;
```

```
err = SslLibCreate(theLibRef, &lib);

/* Configure to have 'debugCallbackFunction' called for each
 * SSL protocol handshake state change */
cb.callback = debugCallbackFunction;
cb.data = NULL;
err = SslLibSet_InfoInterest(theLibRef, lib,
   sslInfoHandshake);
err = SslLibSet_InfoCallback(theLibRef, lib,(void *)&cb);
```

See Also SslLibGetLong(), SslLibSetLong(), SslLibGetPtr()

SslLibSleep Function

Purpose SSL Library's shared library sleep function.

Declared In SslLib.h

Prototype `Err SslLibSleep (UInt16 libRef)`

Parameters → *libRef*
SSL library reference number, obtained from a call to either SysLibFind() or SysLibLoad(), depending on whether or not the SSL library is already loaded.

Returns

SslLibWake Function

Purpose SSL Library's shared library wake function.

Declared In SslLib.h

Prototype `Err SslLibWake (UInt16 libRef)`

Parameters → *libRef*
SSL library reference number, obtained from a call to either SysLibFind() or SysLibLoad(), depending on whether or not the SSL library is already loaded.

Returns

SslOpen Function

Purpose Initializes the passed SslContext.

Declared In `SslLib.h`

Prototype

```
Err SslOpen (UInt16 libRef, SslContext *ctx,
    UInt16 mode, UInt32 timeout)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

↔ *ctx*
The SslContext to start a SSL Handshake with.

→ *mode*
How we should 'start' this connection. A combination of the values listed under "SSL Open Mode Flags" on page 2202.

→ *timeout*
Optional timeout (in system ticks) to use for in Net Library.

Returns Returns `errNone` if the operation completed successfully. Otherwise, this function returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments Depending on the mode, `SslOpen()` may or may not actually send the handshake messages during this function call, but may delay the handshake until the first `SslSend()`/`SslReceive()`. It may not even use the SSL protocol. The mode values can be logically ORed together, and their values affect the functionality of the `SslOpen()` call. If non-fatal network errors occur (timeouts), the function can be re-called. When the function finally returns `errNone`, the SSL handshake will have completed and any `SslContext` will be able to return a valid `SslSession` structure. If one re-calls `SslOpen()`, make sure to not have either of `sslOpenModeClear` or `SslOpenModeSsl` set or the connection will be reset.

Quite a few of the mode parameters set flags against the `SslContext`.

Returns `SslClose()`, `SslContextSetLong()`, `SslContextSetPtr()`

SslPeek Function

Purpose Obtains a pointer into the buffered data that is located in the SslContext.

Declared In `SslLib.h`

Prototype

```
Err SslPeek (UInt16 libRef, SslContext *ctx,
    void **buffer_ptr, Int32 *availableBytes,
    Int32 readSize)
```

Parameters

→ *libRef*
: SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ctx*
: The SslContext to operate on.

← *buffer_ptr*
: The location to place the data pointer.

← *availableBytes*
: The location to place the number of bytes available in **buffer_ptr*.

→ *readSize*
: The maximum number of bytes to return.

Returns Returns `errNone` if the operation completed successfully. Otherwise, this function returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments This function returns a pointer to available data bytes and assigns the number available to *availableBytes*. This function does not copy any data bytes from the `SslContext`, rather it returns a pointer into the buffered data that is located in the `SslContext`. If there were no data bytes in the `SslContext`, data will be read from the network until there are data bytes available. Repeated calls to `SslPeek()` will return the same *buffer_ptr* value until a `SslConsume()` call is done to indicate that bytes no longer need to be buffered. *readSize* is the maximum number of available bytes that will be reported as being available.

This is a more advanced function but is used internally, along with `SslConsume()`, to build the `SslRead()` and `SslReceive()` functions. Its main use is for 'streaming' input data where the

application does not need to allocate it's own data storage bufferes since it can read directly from the `SslContext` buffers. Once a quantity of data is reported as availale in `*availableBytes`, the total will not increase until that number of bytes has been 'consumed'.

See Also `SslConsume()`, `SslRead()`

SslRead Function

Purpose Receives data.

Declared In `SslLib.h`

Prototype

```
Int32 SslRead (UInt16 libRef, SslContext *ctx,
    void *buffer, Int32 bufferLen, Err *errRet)
```

Parameters

→ *libRef*
: SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ctx*
: The SslContext to read from.

← *buffer*
: Buffer into which read data will be placed.

→ *bufferLen*
: Size of buffer (max bytes read).

← *errRet*
: This will contain an error code if return is -1.

Returns Returns the number of bytes successfully received, or -1 if an error occurred.

Comments Performs the same functionality as `SslReceive()`. This call will use the timeout set earlier against the `SslContext`.

See Also `SslWrite()`, `SslSend()`

SslReceive Function

Purpose Receives data.

Declared In `SslLib.h`

Prototype

```
Int16 SslReceive (UInt16 libRef, SslContext *ctx,
    void *buffer, UInt16 bufferLen, UInt16 flags,
    void *fromAddr, UInt16 fromLen, Int32 timeout,
    Err *errRet)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ctx*
The SslContext to use.

← *buffer*
Buffer into which received data will be placed.

→ *bufferLen*
Size of buffer (max bytes received).

→ *flags*
One or more `netIOFlag...` flags. See the Net Library documentation.

← *fromAddr*
Buffer to hold address of sender (NetSocketAddrType).

↔ *fromLen*
Size of *fromAddr* buffer.

→ *timeout*
Max timeout in system ticks. -1 means wait forever.

← *errRet*
This will contain an error code if return is -1.

Returns Returns the number of bytes successfully received, or -1 if an error occurred.

Comments The function returns either the number of bytes successfully received or -1. If -1, there was an error. In that case, an error code will be deposited at the address given by *errRet*.

See Also `SslSend()`, `NetLibReceive()`

SslSend Function

Purpose Sends data over the network.

Declared In `SslLib.h`

Prototype

```
Int16 SslSend (UInt16 libRef, SslContext *ctx,
    const void *buffer, UInt16 bufferLen,
    UInt16 flags, void *toAddr, UInt16 toLen,
    Int32 timeout, Err *errRet)
```

Parameters

→ *libRef*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ctx*
The SslContext to use.

→ *buffer*
Buffer containing data to send.

→ *bufferLen*
Length, in bytes, of data to send.

→ *flags*
One or more `netIOFlag...` flags. See the Net Library documentation.

→ *toAddr*
Address to send to. See the `NetLibSend()` documentation.

→ *toLen*
Size of *toAddr* buffer

→ *timeout*
Max timeout in system ticks. -1 means wait forever.

← *errRet*
This will contain an error code if return is -1.

Returns Returns the number of bytes successfully sent, or -1 if an error occurred.

Comments This function mirrors the `NetLibSend()` function and has similar arguments and semantics.

The function returns either the number of bytes successfully sent or -1. If -1, there was an error. In that case, an error code will be deposited at the address given by *errRet*. The other parameters

are the same as for `NetLibSend()` and are used when the data bytes are written to the network.

See Also `SslReceive()`, `NetLibSend()`

SslWrite Function

Purpose Sends data over the network.

Declared In `SslLib.h`

Prototype

```
Int32 SslWrite (UInt16 libRef, SslContext *ctx,
    const void *buffer, Int32 bufferLen,
    Err *errRet)
```

Parameters

→ *libRef*
: SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

← *ctx*
: The SslContext to write to.

→ *buffer*
: Buffer containing data to write.

→ *bufferLen*
: Length, in bytes, of data to write.

← *errRet*
: This will contain an error code if return is -1.

Returns Returns the number of bytes successfully sent, or -1 if an error occurred.

Comments Performs the same functionality as `SslSend()`. This call will use the timeout set earlier against the `SslContext`.

See Also `SslRead()`, `SslSend()`

Application-Defined Functions

SslCallbackFunc Function

Purpose A function that the application supplies to SslLib that will be called when specific situations occur during the SSL protocol. The callbacks are specific to the particular SslContext or SslLib they are registered with.

Declared In `SslLib.h`

Prototype

```
Int32 (*SslCallbackFunc) (SslCallback *cb,
    Int32 command, Int32 argi, void *argv)
```

Parameters

→ *cb*
The `SslCallback` structure itself.

→ *command*
A command which specifies the reason for the callback. A single callback structure can be used to handle several different types of SslLib callbacks. In this case, the function must have conditional logic to distinguish between the different commands. The command is used to interpret the remaining two parameters, *argi* and *argv*.

→ *argi*
A command-specific 32-bit integer, normally used to specify more information about the reason for the callback.

↔ *argv*
Pointer to a value that is normally determined by the *command* and/or the *argi* arguments.

Returns Returns `errNone` if the callback command was process without error, or a command-specific error code value otherwise.

Comments An application will supply an `SslCallback` structure to the SslLib library. When SslLib needs to then invoke the callback, the callback function is called with four arguments.

When an `SslCallback` is passed into SslLib, a copy is taken of the structure. This means that the structure passed in can be thought of as a template. It is important to remember that the data field will be copied, so if the object this element points to must be destroyed, additional logic will be required. When a SslContext is created, the

SslCallback structures supplied to the SslLib are copied into the SslContext. This could cause problems if not handled correctly if the data pointed to by the data field is dynamic memory.

There are several general 'commands' that all callbacks should expect to receive; these commands are listed under "SSL Callback Commands" on page 2205. There are normally related to creation and destruction of the structure that holds the callback. If the callback does need to perform any action due to these conditions, return 0.

Example

If a callback returns a non-zero value, the SSL library will treat this as an error and return this value back out to the application. This can be used to implement abort functionality. While in the callback, any SslLib functions can be called to retrieve further information. If an `sslCmdInfoAlert` command is being processed, `SslContextGet_LastAlert()` can be called to retrieve the alert message that was received as shown here:

```
alert=SslContextGet_LastAlert(theLibRef, cb->ssl);
```

The alert values that can be returned are listed under "SSL Server Alerts" on page 2212.

There are two defined callbacks currently used by SslLib: "Info" and "Verify."

The Info Callback

For the Info callback, the *command* parameter is set to `sslCmdInfo`. The *argi*, and *argv* values passed are as follows:

argi	argv type
`sslArgInfoHandshake`	`NULL`
`sslArgInfoAlert`	`NULL`
`sslArgInfoReadBefore`	`SslIoBuf`
`sslArgInfoReadAfter`	`SslIoBuf`
`sslArgInfoWriteBefore`	`SslIoBuf`

argi	argv type
`sslArgInfoWriteAfter`	`SslIoBuf`
`sslArgCert`	`SslExtendedItems` – the certificate sent by the server.

See "Info Callbacks" on page 2207 for more information on these *argi* values.

This wealth of information makes it possible for the application to receive notification of state changes in the SSL protocol, receive any SSL protocol alert messages, and track the I/O operation that the SSL protocol is performing. This callback is primarily intended to aid in debugging applications or to provide visual feedback to the progress of the SSL protocol.

The Verify Callback

For the Verify callback, the *command* parameter is set to `sslCmdVerify`. The *argi*, and *argv* values passed are as follows:

argi	argv type
`sslErrVerifyBadSignature`	`SslVerify`
`sslErrVerifyNoTrustedRoot`	`SslVerify`
`sslErrVerifyNotAfter`	`SslVerify`
`sslErrVerifyNotBefore`	`SslVerify`
`sslErrVerifyConstraintViolation`	`SslVerify`
`sslErrVerifyUnknownCriticalExtension`	`SslVerify`
0	`SslVerify`

During the SSL handshake the server side sends a certificate to the client. This certificate contains the server's public key. SslLib attempts to verify that the certificate is valid. During this certificate verification process, if there are any errors, the Verify callback is called.

The application can, through this callback, override any of the error conditions reported during verification. If there is no Verify callback associated with an SslContext, any errors will immediately be returned to the application.

The Verify callback will be called as each certificate in the certificate chain is verified with any error values encountered passed in *argi* until the certificate is verified. If the certificate verifies ok, the 0 value is passed. This process is repeated for each certificate. This means that even if the certificate chain verifies without an error, the callback will be called once for each certificate (with a 0 *argi* value). If an Info callback is also registered, it would be called once after the certificate chain has been verified with the server's certificate. If there is no verification callback, and an error occurs, the application can 'clear' the error and re-call the relevant SslLib function. The verification will proceed from where it was up to.

The SslVerify Structure

The `SslVerify` structure is defined as follows:

```
typedef struct SslVerify_st {
   SslExtendedItems *certificate;
   SslExtendedItems *fieldItems;   /* Problem field base */
   UInt32 field;                   /* Problem field */
   SslExtendedItems *ex;           /* Extension */
   UInt32 depth;                   /* Certificate depth */
   UInt32 state;                   /* Verification state */
} SslVerify;
```

`certificate` is a pointer to a structure containing the certificate currently being processed. `fieldItems` is a pointer to a structure that contains the data element that is currently causing a problem. `field` is the index into `fieldItems` of the erroneous data element. The `ex` field, if there is an error in extension processing, contains the data element that makes up the X509 extension that just failed. `depth` is the level of the certificate being processed, where 0 is the server's certificate, and higher numbers are certificates being used to chain to a trusted root certificate.

The following #defines represent the possible values of the `state` field. They indicate which section of the certificate verification failed.

```
#define sslVerifyFindParent 1
#define sslVerifySignature 2
#define sslVerifyNotBefore 3
#define sslVerifyNotAfterFindParent 4
#define sslVerifyExtensions 5
#define sslVerifyDone 6
```

`fieldItems` will not always be the same pointer as `certificate`. This is especially true during extension errors. If we have an error in an extension, and the extension has been "decomposed," the "decomposed" elements will be in the `ex` field. The object identifier that identifies the extension "decomposed" in `ex` would be `verify->fieldItems.item[verify->field]`.

The following table lists the elements identified by the `fieldItems` and `field` values for a given verification error.

Verification Error	`fieldItems[field]` contains...
`sslErrCertBadSignature`	The server's certificate, which contains the public key entries.
`SslErrCertNoTrustedRoot`	`NULL`
`SslErrCertNotAfter`	`sslExItemTypeX509`, `asn1FldX509NotAfter`
`SslErrCertNotBefore`	`sslExItemTypeX509`, `asn1FldX509NotBefore`
`sslErrCertConstraintViolation`	`asn1ExItemTypeX509Ex` start location
`sslErrCertUnknownCriticalExtension`	`asn1ExItemTypeX509Ex` start location
`SslErrOk`	`NULL`

For the cases of the returned field being the "`asn1ExItemTypeX509Ex` start location", an explanation is as follows:

A certificate can have zero or more extensions. These extensions specify extra information to be used during evaluation of a

certificate. Each extension consists of an "Object identifier," an optional Boolean "critical extension" flag, and the data bytes. The `fieldItems->item[field]` values in this case points to the `SslExtendedItem` that contains the "Object identifier" for that extension. `field+1` will reference either the optional Boolean field that flags the extension as critical or the data bytes.

```
SslVerify *verify;
SslExtendedItem *oid,*critical,*data;

oid= &(verify->fieldItems[verify->field]);
if (verify->fieldItems[verify->field+1].data_type ==
   asn1Boolean) {
   critical=(verify->fieldItems[verify->field+1]);
   data=(verify->fieldItems[verify->field+2]);
} else {
   critical=NULL;
   data= &(verify->fieldItems[verify->field+1]);
}
/* The data bytes for the 'data' is located at
 * ((UInt8 *)verify->fieldItems)+data->offset;
```

SslLib attempts to interpret only critical extensions, so the critical field should always be present. If a critical extension is not understood, the certificate should be rejected. These callback values allow the application to accept a certificate with critical extensions that the application SslLib does not understand.

SslLib recognizes three extensions at this point in time, taken from the X.509 standard:

2.5.29.15	`KeyUsage`
2.5.29.37	`ExtKeyUsage`
2.5.29.19	`BasicConstraints`

If any of these constraints are flagged as critical, an error will not occur (assuming they are valid).

`BasicConstraints` is the only extension currently verified. It specifies if a certificate can be used for signing other certificates. If the certificate is being used incorrectly, an `sslErrCertConstraintViolation` error will be generated. For

this error, the `ex` field of the `SslVerify` structure will potentially contain

```
asn1ExItemTypeX509ExData, asn1FldX509ExBasicConstraintsCa
asn1ExItemTypeX509ExData, asn1FldX509ExBasicConstraintsPathLenConstraint
```

Note that if these optional fields are not in the certificate, they will not be present in the `SslExtendedItems`. The `PathLenConstraint` will also not contain any data bytes; rather,. the numeric value this field contains will be encoded in the `len` field of the `SslExtendedItem`. If this was not the case, the application would have to learn all about decoding ASN.1 integers. The depth field relates to this certificate.

If this error occurs, the application should not override the error, due to its serious nature.

The SslExtendedItems Structure

The `SslExtendedItems` structure is defined as follows:

```
typedef struct SslExtendedItems_st {
   UInt32 length;
   UInt32 num;
   SslExtendedItem eitem[1];
} SslExtendedItems;
```

The `SslExtendedItems` structure is used to hold a set of `SslExtendedItem` structures. The `eitem` field, while defined as a size of one, is actually large enough to hold `num` entries. The `length` field is the total size of the structure. The structure can be copied by allocating `length` bytes and then copying `length` bytes from the `SslExtendedItems` pointer into the new location. An `SslExtendedItems` structure is used to hold sets of related data elements. A set of such values may contain a RSA public key, a certificate, and a certificate extension all in the same `SslExtendedItems` strucuture.

The SslExtendedItem Structure

Each `SslExtendedItem` belongs to a type that is predefined for each of these objects.

Currently defined values are:

`sslExItemTypeX509`	X.509 Certificate.
`SslExItemTypeRSA`	RSA public key.
`sslExItemTypeRDN`	An X.509 Relative Distinguished Name (RDN). This is a complex way of saying a certificate name. Each certificate contains two names, the Subject of the certificate and the Issuer of the certificate. Both are encoded as RDNs which contain multiple fields.
`sslExItemTypeX509Ex`	X.509 certificates can contain what are called Extensions. A certificate can contain multiple extensions. This type is used to specify extensions.
`SslExItemTypeE509ExData`	Used to group 'decomposed' X.509 extensions.

The `SslExtendedItem` structure is defined as follows:

```
typedef struct SslExtendedItem_st {
   UInt16 type;
   UInt16 field;
   UInt16 dataType;
   UInt16 len;
   UInt32 offset;
} SslExtendedItem;
```

An `SslExtendedItem` strucure is a single data element. The `type` field values (as specified in the preceding table) are used to group related items. A single `SslExtendedItems` structure can contain multiple `SslExtendedItem` structures with different `type` values. In this way a single structure can contain elements referring to both a certificate and an RSA public key.

The `field` field contains a type-specific value that is used to identify the `SslExtendedItem`. The values for this field are defined specifically for each type.

The `dataType` field specifies the encoding type of the data. For the cases being used by SslLib, the value is the ASN.1 encoding type. These values are defined in the `SslLibAsn1.h` header file. They are relevant primarily if the application is attempting to display the data bytes.

The `len` field is the length of the data in the `SslExtendedItem`

The `offset` field is the offset from the start of the parent `SslExtendedItems` to the data field.

To access all the data bytes in an `SslExtendedItems` structure:

```
SslExtendedItems *ei;
UInt16 i,j;
UInt8 *p;

for (i=0; i<ei->num; i++) {
   p=((UInt8 *)ei)+ei->eitem[i].offset;

   for (j=0; j<ei->eitem[i].len; j++)
      doSomething(p[j]);
}
```

An `SslExtendedItems` structure will often contain multiple types. When SslLib returns an `SslExtendedItems` structure for a certificate, it will usually contain the types `sslExItemTypeX509`, `sslExItemTypeRSA`, and `sslExItemTypeRDN` (for the subject name).

Example Following is an example of usage from the SslLib, for the 'information' callback:

```
Int32 info_callback(SslCallback *cb,Int32 command,
   Int32 argi,void *argv);

SslCallback infoCB;

infoCB.callback=info_callback;
SslContextSet_InfoCallback(theLibRef, ssl,&infoCB);
SslContextSet_InfoInterest(theLibRef, ssl,
   sslFlgInfoAlert|
   sslFlgInfoHandshake|
   sslFlgInfoIo);

/* We have now configured the SslContext so that
```

```
 * info_callback will be called when 'interesting' events
 * occur. */

Int32 info_callback(SslCallback *cb,Int32 command,
   Int32 argi, void *argv)
   {
   UInt32 alert;

   switch (command){
   case sslCmdInfo:
      /* We have received an 'info' call */
      switch (argi){
      case sslArgInfoHandshake:
         /* The SslContext is in the handshake stage
            of connection establishment. */
         break;
      case sslArgInfoAlert:
         /* An Alert message was received by the SslContext*/
         alert=SslContextGet_LastAlert(theLibRef, cb->ssl);
         break;
      case sslArgInfoReadBefore:
      case sslArgInfoReadAfter:
      case sslArgInfoWriteBefore:
      case sslArgInfoWriteAfter:
         /* The SslContext is doing network operations */
         break;
      }
      break;
   case sslCmdNew:   /* Called when we are 'copied in' */
   case sslCmdFree:   /* Called when we are 'finished' */
   case sslCmdReset:   /* Called instead of 'sslCmdNew' to
      reset the settings */
   case sslCmdSet:   /* Set a value */
   case sslCmdGet:   /* Get a value */
      break;
   }
   return(0);
}
```

87

SSL Library Macros

This chapter provides reference documentation for the macros that your application uses to set and get SslLib and SslContext attribute values. As well, the constants representing those attributes are listed in this chapter.

This chapter is divided into the following sections:

SSL Library Macro Constants 2251
SSL Library Macros 2254

The header file `SslLibMac.h` declares the API that this chapter describes.

Documentation for the functions that these macros employ can be found in Chapter 86, "SSL Library," on page 2189. A detailed description of each attribute can be found under "Attributes" on page 206.

SSL Library Macro Constants

Attribute Values

Purpose These constants represent the attributes that you can set or get by calling `SslContextGet...()`, `SslContextSet...()`, `SslLibGet...()`, or `SslLibSet...()`. The macros documented in "SSL Library Macros" on page 2254 call the appropriate function and supply the proper attribute value. Accordingly, applications should use these macros instead of calling the set or get functions directly.

Complete documentation for the attributes that these values represent can be found in Chapter 8, "Secure Sockets Layer (SSL)," on page 203.

Declared In SslLibMac.h

Constants

```
#define sslAttrAppInt32 0x0F0A0A13
#define sslAttrAppPtr 0x0F090911
#define sslAttrAutoFlush 0x0F070712
#define sslAttrBufferedReuse 0x0F080812
#define sslAttrCertPeerCert 0x0100FF01
#define sslAttrCertPeerCommonName 0x0180FF01
#define sslAttrCertSslVerify 0x0101FF04
#define sslAttrClientCertRequest 0x0F20FF12
#define sslAttrCompat 0x0F010113
#define sslAttrCspCipherSuite 0x0001FF01
#define sslAttrCspCipherSuiteInfo 0x0082FF01
#define sslAttrCspCipherSuites 0x00008101
#define sslAttrCspSslSession 0x00808001
#define sslAttrDontSendShutdown 0x0F212112
#define sslAttrDontWaitForShutdown 0x0F222212
#define sslAttrError 0x0F111113
#define sslAttrErrorState 0x0F05FF14
#define sslAttrHsState 0x0F12FF13
#define sslAttrInfoCallback 0x0F0E0E15
#define sslAttrInfoInterest 0x0F020213
#define sslAttrIoFlags 0x04030303
#define sslAttrIoSocket 0x04010103
#define sslAttrIoStruct 0x04008001
#define sslAttrIoTimeout 0x04020203
#define sslAttrLastAlert 0x0F131313
#define sslAttrLastApi 0x0F1FFF12
#define sslAttrLastIo 0x0F1EFF12
```

```
#define sslAttrLibAppInt32 0x0F0F0F03
#define sslAttrLibAppPtr 0x0F0E0E01
#define sslAttrLibAutoFlush 0x0F0C0C02
#define sslAttrLibBufferedReuse 0x0F0D0D02
#define sslAttrLibCompat 0x0F010103
#define sslAttrLibDontSendShutdown 0x0F121202
#define sslAttrLibDontWaitForShutdown 0x0F131302
#define sslAttrLibInfoCallback 0x0F080805
#define sslAttrLibInfoInterest 0x0F020203
#define sslAttrLibMode 0x0F040403
#define sslAttrLibProtocolVersion 0x0F030303
#define sslAttrLibRbufSize 0x0F101003
#define sslAttrLibReadStreaming 0x0F0B0B02
#define sslAttrLibVerifyCallback 0x0F0A0A05
#define sslAttrLibWbufSize 0x0F111103
#define sslAttrMode 0x0F048013
#define sslAttrProtocolVersion 0x0F030313
#define sslAttrRbufSize 0x0F1B8113
#define sslAttrReadBufPending 0x0F16FF13
#define sslAttrReadOutstanding 0x0F18FF13
#define sslAttrReadRecPending 0x0F17FF13
#define sslAttrReadStreaming 0x0F060612
#define sslAttrSessionReused 0x0F14FF12
#define sslAttrStreaming 0x0F1DFF12
#define sslAttrVerifyCallback 0x0F101015
#define sslAttrWbufSize 0x0F1C8213
#define sslAttrWriteBufPending 0x0F15FF13
```

SSL Library Macros

SslContextGet_AppInt32 Macro

Purpose Obtain the value of the AppInt32 attribute, an arbitrary 32-bit value that an application can attach to an SslContext.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_AppInt32 (refnum, ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either SysLibFind() or SysLibLoad(), depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns Returns the 32-bit value.

Comments SslContextDestroy() does not modify this attribute, so if the data pointed to by this attribute needs to be disposed of, the application must do this itself.

See Also "AppInt32" on page 213, SslContextSet_AppInt32(), SslLibGet_AppInt32()

SslContextGet_AppPtr Macro

Purpose Obtain the value of the AppPtr attribute, an arbitrary pointer value that an application can attach to an SslContext.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_AppPtr (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either SysLibFind() or SysLibLoad(), depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

→ *v*
The address of a pointer variable into which the `AppPtr` attribute value is written.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments `SslContextDestroy()` does not modify this attribute, so if the data pointed to by this attribute needs to be disposed of, the application must do this itself.

See Also "AppPtr" on page 214, `SslContextSet_AppPtr()`, `SslLibGet_AppPtr()`

SslContextGet_AutoFlush Macro

Purpose Determine whether `SslSend()` and `SslWrite()` attempt to immediately send the supplied data bytes to the network.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_AutoFlush (refnum, ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns Returns 0 if data is buffered, or 1 if the data is sent immediately.

Comments It is very important to remember to use `SslFlush()` when AutoFlush is disabled.

See Also "AutoFlush" on page 207, `SslContextSet_AutoFlush()`, `SslLibGet_AutoFlush()`

SslContextGet_BufferedReuse Macro

Purpose Determine if the last message in an SslSession-reused handshake should be buffered instead of being sent over the network.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_BufferedReuse (refnum, ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns Returns 0 if the last message is *not* buffered, or a non-zero value if it is.

See Also "BufferedReuse" on page 222, `SslContextSet_BufferedReuse()`, `SslLibGet_BufferedReuse()`

SslContextGet_CipherSuite Macro

Purpose Identify the cipher suite being used by the current connection.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_CipherSuite (refnum, ssl,
    v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

→ *v*
Pointer to a pointer variable; upon return the pointer variable points to two bytes which identify the cipher suite being used by the current connection. If these two bytes are both set to zero, no cipher suite is being used. Otherwise, see "Cipher Suites" on page 2207 for the list of possible cipher suites.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "CipherSuite" on page 214

SslContextGet_CipherSuiteInfo Macro

Purpose Obtain information relevant to the current cipher suite. This macro populates a `SslCipherSuiteInfo` structure that must have been allocated by the caller.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_CipherSuiteInfo (refnum,
    ssl, v)
```

Parameters

→ *refnum*
: SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
: The SslContext from which the value is to be retrieved.

→ *v*
: Pointer to the `SslCipherSuiteInfo` structure that is to be filled in.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "CipherSuiteInfo" on page 215, `SslContextGet_CipherSuite()`

SslContextGet_CipherSuites Macro

Purpose Obtain the list of cipher suites that the SSL protocol is attempting to use.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_CipherSuites (refnum, ssl,
    v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

→ *v*
Supply the address of a pointer variable; upon return the pointer variable will point to a series of bytes, where the first two bytes indicate the number of bytes that follow, and each pair of bytes after that is one of the values listed under "Cipher Suites" on page 2207. See "CipherSuites" on page 208 for more details on this attribute.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "CipherSuites" on page 208, `SslContextSet_CipherSuites()`, `SslLibGet_CipherSuites()`

SslContextGet_ClientCertRequest Macro

Purpose Determine whether or not the SSL server requested a client certificate.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_ClientCertRequest (refnum,
    ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns Returns 0 if the server did not request a client certificate, or 1 if the server did request one.

See Also "ClientCertRequest" on page 215

SslContextGet_Compat Macro

Purpose Determine which SSL protocol compatibility flags are set. These flags enable compatiblity with certain incorrect SSL protocol implementations.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_Compat (refnum, ssl)
```

Parameters → *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns Returns a 32-bit integer value that is the logical OR of those compatibility flags that have been set. See "Compatibility Flags" on page 2204 for the defined constants that correspond to the compatibility flags.

See Also "Compat" on page 215, `SslContextSet_Compat()`, `SslLibGet_Compat()`

SslContextGet_DontSendShutdown Macro

Purpose Determine whether or not an `SslClose()` will send a shutdown message to the server.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_DontSendShutdown (refnum,
    ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns Returns zero if `SslClose()` does send a shutdown message to the server, or a non-zero value if it doesn't.

See Also "DontSendShutdown" on page 222, `SslContextSet_DontSendShutdown()`, `SslLibGet_DontSendShutdown()`, `SslContextGet_DontWaitForShutdown()`

SslContextGet_DontWaitForShutdown Macro

Purpose Determine whether or not the SslContext will wait for a shutdown message in `SslClose()`.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_DontWaitForShutdown
    (refnum, ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns Returns zero if the SslContext waits for a shutdown message, or a non-zero value if it doesn't.

See Also "DontWaitForShutdown" on page 223, `SslLibGet_DontWaitForShutdown()`, `SslContextGet_DontSendShutdown()`, `SslContextSet_DontWaitForShutdown()`

SslContextGet_Error Macro

Purpose Obtain the error value produced when a fatal error occurs while using an SslContext.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_Error (refnum, ssl)
```

Parameters

→ *refnum*
: SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
: The SslContext from which the value is to be retrieved.

Returns Returns 0 if there was no error, or one of the error codes listed under "SSL Library Errors" on page 2214 if an error did occur.

Comments Once the error attribute is set, the SslLib network APIs will continue to return this error (unless the error is a non-fatal error) until either an SSL Reset is performed on the SslContext or the error is cleared.

See Also "Error" on page 209, `SslContextSet_Error()`

SslContextGet_HsState Macro

Purpose Determine the state that the SSL protocol is currently in.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_HsState (refnum, ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns One of the values are defined under "SSL Protocol States" on page 2210 to indicate the SSL protocol handshake.

See Also "HsState" on page 216

SslContextGet_InfoCallback Macro

Purpose Obtain a pointer to the callback function called when various situations occur during the usage of an SslContext. This callback is primarily intended for debugging and feedback purposes.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_InfoCallback (refnum, ssl,
    v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

→ *v*
Pass the address of a pointer variable; upon return this variable will point to the callback function. The callback function is of type `SslCallbackFunc()`.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments See "InfoCallback" on page 216 and `SslCallbackFunc()` (documented on page 2241) for more on how this callback function is used.

See Also "InfoCallback" on page 216, `SslContextSet_InfoCallback()`, `SslLibGet_InfoCallback()`

SslContextGet_InfoInterest Macro

Purpose Obtain the flags that specify the events for which the Info callback will be called.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_InfoInterest (refnum, ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns The logical ORing of the `sslFlgInfo...` values listed under "InfoInterest Values" on page 2209.

See Also "InfoInterest" on page 216, `SslContextSet_InfoInterest()`, `SslLibGet_InfoInterest()`

SslContextGet_IoFlags Macro

Purpose Obtain the flags value that is passed to Net Library calls.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_IoFlags (refnum, ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns The 32-bit flags value that is passed to Net Library calls.

Comments Since we will normally be using TCP connections with SSL, this attribute is more included for completeness rather than utility. Read about this flags value in the `NetLibSend()` and `NetLibReceive()` documentation.

NOTE: The `netIOFlagOutOfBand` and `netIOFlagPeek` values are not valid and will be silently removed.

See Also "IoFlags" on page 217, `SslContextSet_IoFlags()`

SslContextGet_IoStruct Macro

Purpose Obtain the SslContext's internal `SslSocket` structure.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_IoStruct (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *v*
Pass the address of a pointer variable. Upon return the variable will point to the SslContext's `SslSocket` structure.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "IoStruct" on page 217, `SslContextSet_IoStruct()`

SslContextGet_IoTimeout Macro

Purpose Obtain the SslContext's internal timeout value.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_IoTimeout (refnum, ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns The internal timeout value, in seconds.

Comments When a call is made into the SslLib API which does not specify a timeout, this internal value is used. If the API call has a timeout value, it overrides this internal value.

By default, the SslContext's internal timeout value is 10 seconds.

See Also "IoTimeout" on page 217, `SslContextSet_IoTimeout()`

SslContextGet_LastAlert Macro

Purpose Obtain the last alert value received from the server.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_LastAlert (refnum, ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns One of the values listed under "SSL Server Alerts" on page 2212.

Comments Non-fatal alerts have a value of the form `0x01`*XX*, while fatal alerts have the form `0x02`*XX*. SslLib will fail on fatal alerts and continue on non-fatal alerts.

See Also "LastAlert" on page 218, `SslContextSet_LastAlert()`

SslContextGet_LastApi Macro

Purpose Identify the last SslLib API call that was made.

Declared In `SslLibMac.h`

Prototype `#define SslContextGet_LastApi (refnum, ssl)`

Parameters → *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns Returns one of the values listed under "LastApi Attribute Values" on page 2209.

See Also "LastApi" on page 218

SslContextGet_LastIo Macro

Purpose Identify the last network operation performed.

Declared In `SslLibMac.h`

Prototype `#define SslContextGet_LastIo (refnum, ssl)`

Parameters → *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns Returns one of the values listed under "LastIO Attribute Values" on page 2210.

See Also "LastIo" on page 218

SslContextGet_Mode Macro

Purpose Obtain the value of the Mode attribute, which controls whether the SSL protocol is on or off.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_Mode (refnum, ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns Returns one of the values listed under "Mode Attribute Values" on page 2203.

See Also "Mode" on page 210, `SslContextSet_Mode()`, `SslLibGet_Mode()`

SslContextGet_PeerCert Macro

Purpose Obtain the certificate supplied by the other end of the SSL connection, if one is available.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_PeerCert (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

→ *v*
Pass the address of a pointer variable. Upon return that variable will reference an `SslExtendedItems` data structure internal to the SslContext that contains the peer certificate.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments The returned pointer references an `SslExtendedItems` data structure which is internal to the SslContext and will be disposed of by the SslContext. If a new connection is established with the SslContext, previously returned `PeerCert` pointers will become invalid. If the application wishes to preserve the certificate for an extended period, it should make a local copy.

See Also "PeerCert" on page 219, "The SslExtendedItems Structure" on page 2247, `SslContextGet_PeerCommonName()`

SslContextGet_PeerCommonName Macro

Purpose Obtain the server certificate's common name.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_PeerCommonName (refnum,
    ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

→ *v*
Supply the address of a pointer variable. Upon return the variable will point to an `SslExtendedItems` structure containing the common name for the server's certificate (or, if there is no common name available, the variable will be set to `NULL`.)

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Example The following code shows how to access the common name from within the `SslExtendedItem` structure:

```
SslExtendedItems *cert;
SslExtendedItem *commonName;
UInt16 length;
UInt8 *bytes;

SslContextGet_PeerCert(ssl, &cert);
if (cert == NULL) goto err;
SslContextGet_PeerCommonName(ssl,&commonName);
length=commonName->len;
bytes=((Int8 *)cert)+commonName->offset;
// bytes now points to the common name, and length contains
// the length of the common name string.
```

See Also "PeerCommonName" on page 219, "The SslExtendedItems Structure" on page 2247

SslContextGet_ProtocolVersion Macro

Purpose Determine which version of the SSL protocol is being used.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_ProtocolVersion (refnum,
    ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns One of the values listed under "Protocol Versions" on page 2204.

See Also "ProtocolVersion" on page 220, `SslContextSet_ProtocolVersion()`, `SslLibGet_ProtocolVersion()`

SslContextGet_RbufSize Macro

Purpose Obtain the size, in bytes, of the SslContext's read buffer.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_RbufSize (refnum, ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns The read buffer size. This is a value that ranges from 0 to 16384.

See Also "RbufSize" on page 211, `SslContextSet_RbufSize()`, `SslLibGet_RbufSize()`

SslContextGet_ReadBufPending Macro

Purpose Obtains the number of data bytes that are currently buffered for reading from the SslContext.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_ReadBufPending (refnum,
    ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns The number of buffered read bytes.

Comments The returned number of bytes also includes bytes used for encoding SSL records.

See Also "ReadBufPending" on page 223

SslContextGet_ReadOutstanding Macro

Purpose Obtain the number of bytes in the current record that have not been read from the network

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_ReadOutstanding (refnum,
    ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns The number of bytes outstanding in current SSL record.

See Also "ReadOutstanding" on page 224

SslContextGet_ReadRecPending Macro

Purpose Obtain the number of application data bytes that are buffered, awaiting the application to read.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_ReadRecPending (refnum,
    ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns The number of buffered SSL record bytes.

See Also "ReadRecPending" on page 224, `SslContextGet_ReadOutstanding()`

SslContextGet_ReadStreaming Macro

Purpose Determine whether data can be returned to the application from the SSL connection before the full record has been downloaded.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_ReadStreaming (refnum, ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns Returns a zero value if data is only returned when the full record has been downloaded, or a non-zero value otherwise.

See Also "ReadStreaming" on page 224, `SslContextSet_ReadStreaming()`, `SslLibGet_ReadStreaming()`

SslContextGet_SessionReused Macro

Purpose Determine whether the SSL handshake was able to perform a truncated handshake by re-using the SSL session values in the SslContext.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_SessionReused (refnum, ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns Returns a non-zero value if the SSL handshake was able to perform a truncated handshake by re-using the SSL session values in the SslContext, or zero if it wasn't.

See Also "SessionReused" on page 220

SslContextGet_Socket Macro

Purpose Obtain the Net Library socket that the SslContext should use to perform its network I/O operations.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_Socket (refnum, ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns The `NetSocketRef` for the SslContext's Net Library socket.

See Also "Socket" on page 212, `SslContextSet_Socket()`

SslContextGet_SslSession Macro

Purpose Get the SslContext's `SslSession` structure. This is either the `SslSession` currently being used, or the `SslSession` for this SslContext to use to establish its next connection.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_SslSession (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

→ *v*
Pass the address of a pointer variable; upon return this variable will point to the SslContext's `SslSession` structure.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "SslSession" on page 221, `SslContextSet_SslSession()`

SslContextGet_SslVerify Macro

Purpose Obtain a pointer to the structure containing the verification state. This structure can be helpful when attempting to resolve any problems that SslLib may have encountered during certificate verifcation.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_SslVerify (refnum, ssl, v)
```

Parameters

→ *refnum*
: SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
: The SslContext from which the value is to be retrieved.

→ *v*
: Supply the address of a pointer variable. Upon return the variable will point to an `SslVerify` structure containing the preserved state. See "The SslVerify Structure" on page 2244 for a description of this structure.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "SslVerify" on page 221

SslContextGet_Streaming Macro

Purpose Determine if the current SslContext is doing read-streaming.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_Streaming (refnum, ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns Returns 1 if the current SslContext is doing read-streaming.

See Also "Streaming" on page 225

SslContextGet_VerifyCallback Macro

Purpose Obtain a pointer to the callback function used to assist with certificate verification.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_VerifyCallback (refnum,
    ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

→ *v*
Pass the address of a pointer variable; upon return this variable will point to the callback function. The callback function is of type `SslCallbackFunc()`.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments See "VerifyCallback" on page 212 and `SslCallbackFunc()` (documented on page 2241) for more on how this callback function is used to verify a certificate.

See Also "VerifyCallback" on page 212, `SslContextSet_VerifyCallback()`, `SslLibGet_VerifyCallback()`

SslContextGet_WbufSize Macro

Purpose Obtain the size, in bytes, of the SslContext's write buffer.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_WbufSize (refnum, ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns The write buffer size. This is a value that ranges from 0 to 16384.

See Also "WbufSize" on page 213, `SslContextSet_WbufSize()`, `SslLibGet_WbufSize()`

SslContextGet_WriteBufPending Macro

Purpose Obtains the number of bytes in the SslContext's write buffer waiting to be sent to the remote machine.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextGet_WriteBufPending (refnum,
    ssl)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext from which the value is to be retrieved.

Returns The number of buffered bytes waiting to be sent.

Comments This value will normally be zero unless **AutoFlush** is disabled and/or non-blocking I/O is being used.

See Also "WriteBufPending" on page 225, `SslContextGet_AutoFlush()`

SslContextSet_AppInt32 Macro

Purpose Set the value of the `AppInt32` attribute, an arbitrary 32-bit value that an application can attach to an SslContext.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_AppInt32 (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
The 32-bit value.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments `SslContextDestroy()` does not modify this attribute, so if the data pointed to by this attribute needs to be disposed of, the application must do this itself.

See Also "AppInt32" on page 213, `SslContextGet_AppInt32()`, `SslLibSet_AppInt32()`

SslContextSet_AppPtr Macro

Purpose Set the value of the AppPtr attribute, an arbitrary pointer value that an application can attach to an SslContext.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_AppPtr (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either SysLibFind() or SysLibLoad(), depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
The AppPtr attribute value.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments SslContextDestroy() does not modify this attribute, so if the data pointed to by this attribute needs to be disposed of, the application must do this itself.

See Also "AppPtr" on page 214, SslContextGet_AppPtr(), SslLibSet_AppPtr()

SslContextSet_AutoFlush Macro

Purpose Specify whether SslSend() and SslWrite() should attempt to immediately send the supplied data bytes to the network.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_AutoFlush (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either SysLibFind() or SysLibLoad(), depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
Pass 0 to have the data buffered, or 1 to have the data sent immediately.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments It is very important to remember to use `SslFlush()` when AutoFlush is disabled.

See Also "AutoFlush" on page 207, `SslContextGet_AutoFlush()`, `SslLibSet_AutoFlush()`

SslContextSet_BufferedReuse Macro

Purpose Specify whether the last message in an SslSession-reused handshake should be buffered instead of being sent over the network.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_BufferedReuse (refnum, ssl,
    v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
Supply a value of 0 if the last message should *not* be buffered, or a non-zero value if it should.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "BufferedReuse" on page 222, `SslContextGet_BufferedReuse()`, `SslLibSet_BufferedReuse()`

SslContextSet_CipherSuites Macro

Purpose Specify the list of cipher suites that the SSL protocol should attempt to use.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_CipherSuites (refnum, ssl,
    v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
Pointer to an array of byte pairs. The first two bytes contain the number of bytes that follow, and each successive pair is one of the values listed under "Cipher Suites" on page 2207.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "CipherSuites" on page 208, `SslContextGet_CipherSuites()`, `SslLibSet_CipherSuites()`

SslContextSet_Compat Macro

Purpose Determine how compatible the SSL protocol should be with certain incorrect SSL protocol implementations.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_Compat (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*

The logical OR of those compatibility flags that correspond to the SSL protocol incompatibilities that should be accommodated. See "Compatibility Flags" on page 2204 for the defined constants that correspond to the compatibility flags.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "Compat" on page 215, `SslContextGet_Compat()`, `SslLibSet_Compat()`

SslContextSet_DontSendShutdown Macro

Purpose Specify whether or not an `SslClose()` should send a shutdown message to the server.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_DontSendShutdown (refnum,
    ssl, v)
```

Parameters

→ *refnum*

SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*

The SslContext on which to operate.

→ *v*

Zero if `SslClose()` should send a shutdown message, or a non-zero value if it shouldn't.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "DontSendShutdown" on page 222, `SslContextGet_DontSendShutdown()`, `SslLibSet_DontSendShutdown()`, `SslContextSet_DontWaitForShutdown()`

SslContextSet_DontWaitForShutdown Macro

Purpose Specify whether or not the SslContext should wait for a shutdown message in `SslClose()`.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_DontWaitForShutdown
    (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
Zero if the SslContext waits for a shutdown message, or a non-zero value if it doesn't.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "DontWaitForShutdown" on page 223, `SslContextGet_DontWaitForShutdown()`, `SslContextSet_DontSendShutdown()`, `SslLibSet_DontWaitForShutdown()`

SslContextSet_Error Macro

Purpose Associate a new error value with an SslContext. This value overrides any error value produced when a fatal error occurs while using an SslContext.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_Error (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
The new error value. A value of 0 corresponds to "no error."

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments Once the error attribute is set, the SslLib network APIs will continue to return this error (unless the error is a non-fatal error) until either an SSL Reset is performed on the SslContext or the error is cleared.

See Also "Error" on page 209, `SslContextGet_Error()`

SslContextSet_InfoCallback Macro

Purpose Specify the callback function called when various situations occur during the usage of an SslContext. This callback is primarily intended for debugging and feedback purposes.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_InfoCallback (refnum, ssl,
    v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
A pointer to a callback function of type `SslCallbackFunc()`.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments See "InfoCallback" on page 216 and `SslCallbackFunc()` (documented on page 2241) for more on how this callback function is used.

See Also "InfoCallback" on page 216, `SslContextGet_InfoCallback()`, `SslLibSet_InfoCallback()`

SslContextSet_InfoInterest Macro

Purpose Specify the events for which the Info callback will be called.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_InfoInterest (refnum, ssl,
    v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
The logical ORing of the `sslFlgInfo...` values listed under "InfoInterest Values" on page 2209.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "InfoInterest" on page 216, `SslContextGet_InfoInterest()`, `SslLibSet_InfoInterest()`

SslContextSet_IoFlags Macro

Purpose Specify the flags value that is to be passed to Net Library calls.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_IoFlags (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
The 32-bit flags value to be passed to Net Library calls.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments Since we will normally be using TCP connections with SSL, this attribute is more included for completeness rather than utility. Read about this flags value in the `NetLibSend()` and `NetLibReceive()` documentation.

NOTE: The `netIOFlagOutOfBand` and `netIOFlagPeek` values are not valid and will be silently removed.

See Also "IoFlags" on page 217, `SslContextGet_IoFlags()`

SslContextSet_IoStruct Macro

Purpose Specify the `SslSocket` structure to be used by an SslContext.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_IoStruct (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
Pointer to the `SslSocket` structure.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "IoStruct" on page 217, `SslContextSet_IoStruct()`

SslContextSet_IoTimeout Macro

Purpose Specify the SslContext's internal timeout value.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_IoTimeout (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
The new internal timeout value, in seconds.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments When a call is made into the SslLib API which does not specify a timeout, this internal value is used. If the API call has a timeout value, it overrides this internal value.

By default, the SslContext's internal timeout value is 10 seconds.

See Also "IoTimeout" on page 217, `SslContextGet_IoTimeout()`

SslContextSet_LastAlert Macro

Purpose Override the last alert value received from the server.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_LastAlert (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
The new "last alert" value. This should be one of the values listed under "SSL Server Alerts" on page 2212.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments Non-fatal alerts have a value of the form `0x01`*XX*, while fatal alerts have the form `0x02`*XX*. SslLib will fail on fatal alerts and continue on non-fatal alerts.

See Also "LastAlert" on page 218, `SslContextGet_LastAlert()`

SslContextSet_Mode Macro

Purpose Specify the value of the Mode attribute, which controls whether the SSL protocol is on or off.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_Mode (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
One of the values listed under "Mode Attribute Values" on page 2203.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "Mode" on page 210, `SslContextGet_Mode()`, `SslLibSet_Mode()`

SslContextSet_ProtocolVersion Macro

Purpose Specify the version of the SSL protocol to use.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_ProtocolVersion (refnum,
    ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
One of the values listed under "Protocol Versions" on page 2204.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments SslLib implements only SSLv3 at this point in time, so modification of this value is not a good idea.

See Also "ProtocolVersion" on page 220, `SslContextGet_ProtocolVersion()`, `SslLibSet_ProtocolVersion()`

SslContextSet_RbufSize Macro

Purpose Specify the size, in bytes, of the SslContext's read buffer.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_RbufSize (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
The read buffer size. This is a value that ranges from 0 to 16384.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "RbufSize" on page 211, `SslContextGet_RbufSize()`, `SslLibSet_RbufSize()`

SslContextSet_ReadStreaming Macro

Purpose Specify whether data can be returned to the application from the SSL connection before the full record has been downloaded.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_ReadStreaming (refnum, ssl,
    v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
A zero value if data is only to be returned when the full record has been downloaded, or a non-zero value otherwise.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "ReadStreaming" on page 224, `SslContextGet_ReadStreaming()`, `SslLibSet_ReadStreaming()`

SslContextSet_Socket Macro

Purpose Specify the Net Library socket that the SslContext should use to perform its network I/O operations.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_Socket (refnum, ssl, v)
```

Parameters

→ *refnum*
: SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
: The SslContext on which to operate.

→ *v*
: The `NetSocketRef` for the Net Library socket.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "Socket" on page 212, `SslContextGet_Socket()`

SslContextSet_SslSession Macro

Purpose Specify the SslContext's `SslSession` structure. This is the `SslSession` for this SslContext to use to establish its next connection.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_SslSession (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
A pointer to the `SslSession` structure.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "SslSession" on page 221, `SslContextGet_SslSession()`

SslContextSet_VerifyCallback Macro

Purpose Specify the callback function to assist with certificate verification.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_VerifyCallback (refnum,
    ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
A pointer to a callback function of type `SslCallbackFunc()`.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments See "VerifyCallback" on page 212 and `SslCallbackFunc()` (documented on page 2241) for more on how this callback function is used to verify a certificate.

See Also "VerifyCallback" on page 212, `SslContextGet_VerifyCallback()`, `SslLibSet_VerifyCallback()`

SslContextSet_WbufSize Macro

Purpose Specify the size, in bytes, of the SslContext's write buffer.

Declared In `SslLibMac.h`

Prototype

```
#define SslContextSet_WbufSize (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *ssl*
The SslContext on which to operate.

→ *v*
The write buffer size. This is a value that ranges from 0 to 16384.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "WbufSize" on page 213, `SslContextGet_WbufSize()`, `SslLibSet_WbufSize()`

SslLibGet_AppInt32 Macro

Purpose Obtain the value of the `AppInt32` attribute, an arbitrary 32-bit value that an application can attach to an SslLib.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibGet_AppInt32 (refnum, lib)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib from which the value is to be retrieved.

Returns Returns the 32-bit value.

Comments `SslLibDestroy()` does not modify this attribute, so if the data pointed to by this attribute needs to be disposed of, the application must do this itself.

See Also "AppInt32" on page 213, `SslLibSet_AppInt32()`, `SslContextGet_AppInt32()`

SslLibGet_AppPtr Macro

Purpose Obtain the value of the `AppPtr` attribute, an arbitrary pointer value that an application can attach to an SslLib.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibGet_AppPtr (refnum, lib, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib from which the value is to be retrieved.

→ *v*
The address of a pointer variable into which the `AppPtr` attribute value is written.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments `SslLibDestroy()` does not modify this attribute, so if the data pointed to by this attribute needs to be disposed of, the application must do this itself.

See Also "AppPtr" on page 214, `SslLibSet_AppPtr()`, `SslContextGet_AppPtr()`

SslLibGet_AutoFlush Macro

Purpose Determine whether `SslSend()` and `SslWrite()` attempt to immediately send the supplied data bytes to the network.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibGet_AutoFlush (refnum, lib)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib from which the value is to be retrieved.

Returns Returns 0 if data is buffered, or 1 if the data is sent immediately.

Comments It is very important to remember to use `SslFlush()` when AutoFlush is disabled.

See Also "AutoFlush" on page 207, `SslLibSet_AutoFlush()`, `SslContextGet_AutoFlush()`

SslLibGet_BufferedReuse Macro

Purpose Determine if the last message in an SslSession-reused handshake should be buffered instead of being sent over the network.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibGet_BufferedReuse (refnum, lib)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib from which the value is to be retrieved.

Returns Returns 0 if the last message is *not* buffered, or a non-zero value if it is.

See Also "BufferedReuse" on page 222, `SslLibSet_BufferedReuse()`, `SslContextGet_BufferedReuse()`

SslLibGet_CipherSuites Macro

Purpose Obtain the list of cipher suites that the SSL protocol is attempting to use. This list is inherited from the SslLib when an SslContext is created.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibGet_CipherSuites (refnum, ssl, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib from which the value is to be retrieved.

→ *v*
Supply the address of a pointer variable; upon return the pointer variable will point to a series of bytes, where the first two bytes indicate the number of bytes that follow, and each pair of bytes after that is one of the values listed under

"Cipher Suites" on page 2207. See "CipherSuites" on page 208 for more details on this attribute.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "CipherSuites" on page 208, `SslLibSet_CipherSuites()`, `SslContextGet_CipherSuites()`

SslLibGet_Compat Macro

Purpose Determine which SSL protocol compatibility flags are set. These flags enable compatiblity with certain incorrect SSL protocol implementations.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibGet_Compat (refnum, lib)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib from which the value is to be retrieved.

Returns Returns a 32-bit integer value that is the logical OR of those compatibility flags that have been set. See "Compatibility Flags" on page 2204 for the defined constants that correspond to the compatibility flags.

See Also "Compat" on page 215, `SslLibSet_Compat()`, `SslContextGet_Compat()`

SslLibGet_DontSendShutdown Macro

Purpose Determine whether or not an `SslClose()` will send a shutdown message to the server.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibGet_DontSendShutdown (refnum, lib)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib from which the value is to be retrieved.

Returns Returns zero if `SslClose()` does send a shutdown message to the server, or a non-zero value if it doesn't.

See Also "DontSendShutdown" on page 222, `SslLibSet_DontSendShutdown()`, `SslContextGet_DontSendShutdown()`, `SslLibGet_DontWaitForShutdown()`

SslLibGet_DontWaitForShutdown Macro

Purpose Determine whether or not the SslLib will wait for a shutdown message in `SslClose()`.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibGet_DontWaitForShutdown (refnum,
    lib)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib from which the value is to be retrieved.

Returns Returns zero if the SslLib waits for a shutdown message, or a non-zero value if it doesn't.

See Also "DontWaitForShutdown" on page 223, `SslContextGet_DontWaitForShutdown()`, `SslLibGet_DontSendShutdown()`, `SslLibSet_DontWaitForShutdown()`

SslLibGet_InfoCallback Macro

Purpose Obtain a pointer to the callback function called when various situations occur during the usage of an SslLib. This callback is primarily intended for debugging and feedback purposes.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibGet_InfoCallback (refnum, lib, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib from which the value is to be retrieved.

→ *v*
Pass the address of a pointer variable; upon return this variable will point to the callback function. The callback function is of type `SslCallbackFunc()`.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments See "InfoCallback" on page 216 and `SslCallbackFunc()` (documented on page 2241) for more on how this callback function is used.

See Also "InfoCallback" on page 216, `SslLibSet_InfoCallback()`, `SslContextGet_InfoCallback()`

SslLibGet_InfoInterest Macro

Purpose Obtain the flags that specify the events for which the Info callback will be called.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibGet_InfoInterest (refnum, lib)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib from which the value is to be retrieved.

Returns The logical ORing of the `sslFlgInfo...` values listed under "InfoInterest Values" on page 2209.

See Also "InfoInterest" on page 216, `SslLibSet_InfoInterest()`, `SslContextGet_InfoInterest()`

SslLibGet_Mode Macro

Purpose Obtain the value of the Mode attribute, which controls whether the SSL protocol is on or off.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibGet_Mode (refnum, lib)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib from which the value is to be retrieved.

Returns Returns one of the values listed under "Mode Attribute Values" on page 2203.

See Also "Mode" on page 210, `SslLibSet_Mode()`, `SslContextGet_Mode()`

SslLibGet_ProtocolVersion Macro

Purpose Determine which version of the SSL protocol is being used.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibGet_ProtocolVersion (refnum, lib)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib from which the value is to be retrieved.

Returns One of the values listed under "Protocol Versions" on page 2204.

See Also "ProtocolVersion" on page 220, `SslLibSet_ProtocolVersion()`, `SslContextGet_ProtocolVersion()`

SslLibGet_RbufSize Macro

Purpose Obtain the size, in bytes, of the read buffer. Read and write buffers are associated with an SslContext; SslContexts created against this SslLib will inherit the SslLib's buffer values.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibGet_RbufSize (refnum, lib)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib from which the value is to be retrieved.

Returns The read buffer size. This is a value that ranges from 0 to 16384.

See Also "RbufSize" on page 211, `SslLibSet_RbufSize()`, `SslContextGet_RbufSize()`

SslLibGet_ReadStreaming Macro

Purpose Determine whether data can be returned to the application from the SSL connection before the full record has been downloaded.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibGet_ReadStreaming (refnum, lib)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib from which the value is to be retrieved.

Returns Returns a zero value if data is only returned when the full record has been downloaded, or a non-zero value otherwise.

See Also "ReadStreaming" on page 224, `SslContextGet_ReadStreaming()`, `SslLibSet_ReadStreaming()`

SslLibGet_VerifyCallback Macro

Purpose Obtain a pointer to the callback function used to assist with certificate verification.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibGet_VerifyCallback (refnum, lib, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib from which the value is to be retrieved.

→ *v*
Pass the address of a pointer variable; upon return this variable will point to the callback function. The callback function is of type `SslCallbackFunc()`.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments See "VerifyCallback" on page 212 and `SslCallbackFunc()` (documented on page 2241) for more on how this callback function is used to verify a certificate.

See Also "VerifyCallback" on page 212, `SslLibSet_VerifyCallback()`, `SslContextGet_VerifyCallback()`

SslLibGet_WbufSize Macro

Purpose Obtain the size, in bytes, of the write buffer. Read and write buffers are associated with an SslContext; SslContexts created against this SslLib will inherit the SslLib's buffer values.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibGet_WbufSize (refnum, lib)
```

Parameters

→ *refnum*
: SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
: The SslLib from which the value is to be retrieved.

Returns The write buffer size. This is a value that ranges from 0 to 16384.

See Also "WbufSize" on page 213, `SslLibSet_WbufSize()`, `SslContextGet_WbufSize()`

SslLibSet_AppInt32 Macro

Purpose Set the value of the `AppInt32` attribute, an arbitrary 32-bit value that an application can attach to an SslLib.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibSet_AppInt32 (refnum, lib, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib on which to operate.

→ *v*
The 32-bit value.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments `SslLibDestroy()` does not modify this attribute, so if the data pointed to by this attribute needs to be disposed of, the application must do this itself.

See Also "AppInt32" on page 213, `SslLibGet_AppInt32()`, `SslContextSet_AppInt32()`

SslLibSet_AppPtr Macro

Purpose Set the value of the `AppPtr` attribute, an arbitrary pointer value that an application can attach to an SslLib.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibSet_AppPtr (refnum, lib, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib on which to operate.

→ *v*
The `AppPtr` attribute value.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments `SslLibDestroy()` does not modify this attribute, so if the data pointed to by this attribute needs to be disposed of, the application must do this itself.

See Also "AppPtr" on page 214, `SslLibGet_AppPtr()`, `SslContextSet_AppPtr()`

SslLibSet_AutoFlush Macro

Purpose Specify whether `SslSend()` and `SslWrite()` should attempt to immediately send the supplied data bytes to the network.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibSet_AutoFlush (refnum, lib, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib on which to operate.

→ *v*
Pass 0 to have the data buffered, or 1 to have the data sent immediately.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments It is very important to remember to use `SslFlush()` when AutoFlush is disabled.

See Also "AutoFlush" on page 207, `SslContextGet_AutoFlush()`, `SslLibSet_AutoFlush()`

SslLibSet_BufferedReuse Macro

Purpose Specify whether the last message in an SslSession-reused handshake should be buffered instead of being sent over the network.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibSet_BufferedReuse (refnum, lib, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib on which to operate.

→ *v*
Supply a value of 0 if the last message should *not* be buffered, or a non-zero value if it should.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "BufferedReuse" on page 222, `SslLibGet_BufferedReuse()`, `SslContextSet_BufferedReuse()`

SslLibSet_CipherSuites Macro

Purpose Specify the list of cipher suites that the SSL protocol should attempt to use. When an SslContext is created, it inherits this list of cipher suites.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibSet_CipherSuites (refnum, lib, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib on which to operate.

→ *v*

Pointer to an array of byte pairs. The first two bytes contain the number of bytes that follow, and each successive pair is one of the values listed under "Cipher Suites" on page 2207.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "CipherSuites" on page 208, `SslLibGet_CipherSuites()`, `SslContextSet_CipherSuites()`

SslLibSet_Compat Macro

Purpose Determine how compatible the SSL protocol should be with certain incorrect SSL protocol implementations.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibSet_Compat (refnum, lib, v)
```

Parameters

→ *refnum*

SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*

The SslLib on which to operate.

→ *v*

The logical OR of those compatibility flags that correspond to the SSL protocol incompatibilities that should be accommodated. See "Compatibility Flags" on page 2204 for the defined constants that correspond to the compatibility flags.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "Compat" on page 215, `SslLibGet_Compat()`, `SslContextSet_Compat()`

SslLibSet_DontSendShutdown Macro

Purpose Specify whether or not an `SslClose()` should send a shutdown message to the server.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibSet_DontSendShutdown (refnum, lib,
    v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib on which to operate.

→ *v*
Zero if `SslClose()` should send a shutdown message, or a non-zero value if it shouldn't.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "DontSendShutdown" on page 222, `SslLibGet_DontSendShutdown()`, `SslContextSet_DontSendShutdown()`, `SslLibSet_DontWaitForShutdown()`

SslLibSet_DontWaitForShutdown Macro

Purpose Specify whether or not the SSlLib should wait for a shutdown message in `SslClose()`.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibSet_DontWaitForShutdown (refnum,
    lib, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib on which to operate.

→ *v*
Zero if the SslLib waits for a shutdown message, or a non-zero value if it doesn't.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "DontWaitForShutdown" on page 223, `SslLibGet_DontWaitForShutdown()`, `SslLibSet_DontSendShutdown()`, `SslContextSet_DontWaitForShutdown()`

SslLibSet_InfoCallback Macro

Purpose Specify the callback function called when various situations occur during the usage of an SslLib. This callback is primarily intended for debugging and feedback purposes.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibSet_InfoCallback (refnum, lib, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib on which to operate.

→ *v*
A pointer to a callback function of type `SslCallbackFunc()`.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments See "InfoCallback" on page 216 and `SslCallbackFunc()` (documented on page 2241) for more on how this callback function is used.

See Also "InfoCallback" on page 216, `SslLibGet_InfoCallback()`, `SslContextSet_InfoCallback()`

SslLibSet_InfoInterest Macro

Purpose Specify the events for which the Info callback will be called.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibSet_InfoInterest (refnum, lib, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib on which to operate.

→ *v*
The logical ORing of the `sslFlgInfo...` values listed under "InfoInterest Values" on page 2209.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "InfoInterest" on page 216, `SslLibGet_InfoInterest()`, `SslContextSet_InfoInterest()`

SslLibSet_Mode Macro

Purpose Specify the value of the Mode attribute, which controls whether the SSL protocol is on or off.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibSet_Mode (refnum, lib, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib on which to operate.

→ *v*
One of the values listed under "Mode Attribute Values" on page 2203.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "Mode" on page 210, `SslLibGet_Mode()`, `SslContextSet_Mode()`

SslLibSet_ProtocolVersion Macro

Purpose Specify the version of the SSL protocol to use.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibSet_ProtocolVersion (refnum, lib,
    v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib on which to operate.

→ *v*
One of the values listed under "Protocol Versions" on page 2204.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments SslLib implements only SSLv3 at this point in time, so modification of this value is not a good idea.

See Also "ProtocolVersion" on page 220, `SslLibGet_ProtocolVersion()`, `SslContextSet_ProtocolVersion()`

SslLibSet_RbufSize Macro

Purpose Specify the size, in bytes, of the read buffer. Read and write buffers are associated with an SslContext; SslContexts created against this SslLib will inherit the SslLib's buffer values.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibSet_RbufSize (refnum, lib, v)
```

Parameters

→ *refnum*
: SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
: The SslLib on which to operate.

→ *v*
: The read buffer size. This is a value that ranges from 0 to 16384.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "RbufSize" on page 211, `SslLibGet_RbufSize()`, `SslContextSet_RbufSize()`

SslLibSet_ReadStreaming Macro

Purpose Specify whether data can be returned to the application from the SSL connection before the full record has been downloaded.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibSet_ReadStreaming (refnum, lib, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib on which to operate.

→ *v*
A zero value if data is only to be returned when the full record has been downloaded, or a non-zero value otherwise.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "ReadStreaming" on page 224, `SslContextSet_ReadStreaming()`, `SslLibGet_ReadStreaming()`

SslLibSet_VerifyCallback Macro

Purpose Specify the callback function to assist with certificate verification.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibSet_VerifyCallback (refnum, lib, v)
```

Parameters

→ *refnum*
SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
The SslLib on which to operate.

→ *v*
A pointer to a callback function of type `SslCallbackFunc()`.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

Comments See "VerifyCallback" on page 212 and `SslCallbackFunc()` (documented on page 2241) for more on how this callback function is used to verify a certificate.

See Also "VerifyCallback" on page 212, `SslLibGet_VerifyCallback()`, `SslContextSet_VerifyCallback()`

SslLibSet_WbufSize Macro

Purpose Specify the size, in bytes, of the write buffer. Read and write buffers are associated with an SslContext; SslContexts created against this SslLib will inherit the SslLib's buffer values.

Declared In `SslLibMac.h`

Prototype

```
#define SslLibSet_WbufSize (refnum, lib, v)
```

Parameters

→ *refnum*
: SSL library reference number, obtained from a call to either `SysLibFind()` or `SysLibLoad()`, depending on whether or not the SSL library is already loaded.

→ *lib*
: The SslLib on which to operate.

→ *v*
: The write buffer size. This is a value that ranges from 0 to 16384.

Returns Returns `errNone` if the operation completed successfully. Otherwise, the function that this macro evaluates to returns one of the error codes listed under "SSL Library Errors" on page 2214.

See Also "WbufSize" on page 213, `SslLibGet_WbufSize()`, `SslContextSet_WbufSize()`

88

Personal Data Interchange Library

This chapter provides reference material for the Personal Data Interchange (PDI) library, which provides tools for reading and writing vObjects, including vCards and vCals. This chapter discusses the following topics:

- PDI Library Data Structures
- PDI Library Constants
- PDI Library Functions

The header file `PdiLib.h` declares the Personal Data Interchange library API. The header file `PdiConst.h` declares the constants that you use with the PDI library.

For information about how to use the functions and constants described in this chapter, see Chapter 3, "Personal Data Interchange," in *Palm OS Programmer's Companion*, vol. II, *Communications*.

PDI Library Data Structures

This section describes the data structures used with the PDI library functions.

PdiDictionary Typedef

Purpose The `PdiDictionary` type is a simple typedef that represents an internal, binary object.

Prototype

```
typedef UInt8 PdiDictionary;
```

PdiReaderType Struct

Purpose The `PdiReaderType` data structure represents a PDI reader object, which you use to read data from an input stream.

Prototype

```
typedef struct _PdiReader {
  Er error;
  UInt8 encoding;
  CharEncodingType charset;
  UInt16 written;
  UInt8 fieldNum;
  UInt16 property;
  UInt16 propertyValueType;
  UInt16 parameter;
  UInt32 parameterPairs[kPDI_ENTRIES_NUMBER];
  UInt16 customFieldNumber;
  void *appData;
  UInt16 pdiRefNum;
  UInt16 events;
  Char *groupName;
  Char *propertyName;
  Char *parameterName;
  Char *parameterValue;
  Char *propertyValue;
  PdiDictionary *dictionary[2];
} PdiReaderType
```

Fields

`error`
: The most recent error.

`encoding`
: The type of encoding for the property value.

`charset`
: The character set of the property value.

`written`
: The number of characters that have currently been written to the buffer.

`fieldNum`
: The current field number.

`property`
: The ID of the current property.

`propertyValueType`
: The value type of the current property value.

`parameter`
: The ID of the most recently parsed parameter name.

`parameterPairs`
: An integer array with bits set for each parameter value that has been parsed for the current property value.

  You must use the `PdiParameterPairTest` macro to access this field.

`customFieldNumber`
: The number of the custom field parsed by the reader for the current property. Custom fields are used in the Palm™ Address Book.

`appData`
: Application-dependent data field.

`pdiRefNum`
: The library reference number associated with this reader.

`events`
: The mask of events handled by the reader in its most recent operation. This is a combination of some number of the event constants described in Reader Event Constants.

`groupName`
: The group name for the current property.

`propertyName`
: The name of the current property.

`parameterName`
: The name of the current parameter.

`parameterValue`
: The value of the current parameter.

`propertyValue`
: The current property value string.

`dictionary`
: An array of two dictionary pointers. The `dictionary[0]` entry is the main dictionary built into the PDI library, and the `dictionary[1]` entry is an optional, custom dictionary

associated with the reader object with a call to PdiDefineReaderDictionary().

PdiWriterType Struct

Purpose The `PdiWriterType` data structure represents a PDI writer object, which you use to write data to an output stream.

Prototype

```
typedef struct _PdiWriter {
  void *appData;
  UInt16 pdiRefNum;
  UInt16 encoding;
  CharEncodingType charset;
  Err error;
  PdiDictionary *dictionary[2];
} PdiWriterType
```

Fields

`appData`
: Application-dependent data field.

`pdiRefNum`
: The library reference number associated with this reader.

`encoding`
: The type of encoding for the property value.

`charset`
: The character set of the property value.

`error`
: The most recent error.

`dictionary`
: An array of two dictionary pointers. The `dictionary[0]` entry is the main dictionary built into the PDI library, and the `dictionary[1]` entry is an optional, custom dictionary associated with the reader object with a call to PdiDefineReaderDictionary().

PDI Library Constants

This section describes the constants used in the PDI library, which include the following constant types:

- Buffer Management Constants

- Encoding Type Constants
- Error Code Constants
- Parameter Name Constants
- Parameter Value Constants
- Property Name Constants
- Property Type Constants
- Property Value Field Constants
- Property Value Format Constants
- Reader and Writer Options Constants
- Reader Event Constants
- Value Type Constants

Buffer Management Constants

You use the buffer management constants to determine how buffers are managed in the PDI reader.

Constant	Value	Description
`kPdiResizableBuffer`	`0xFFFF`	Indicates that the buffer can be automatically resized by the PDI library.
`kPdiDefaultBufferMaxSize`	`0x3FFF`	The default maximum buffer size, in bytes. You can change the maximum size of a reader's buffer by calling the `PdiDefineResizing()` function.
`kPdiDefaultBufferDeltaSize`	`0x0010`	The default number of bytes by which the input buffer is grown when the PDI library performs automatic resizing. You can change the delta amount of a reader's buffer by calling `PdiDefineResizing()` function.

Encoding Type Constants

You use the encoding type constants to specify the type of encoding used in a vObject reader or writer.

Constant	Value	Description
`kPdiASCIIEncoding`	`0`	The vObject is not encoded.
`kPdiQPEncoding`	`kPdiPAV_ENCODING_QUOTED_PRINTABLE`	The vObject uses the quoted printable encoding.
`kPdiB64Encoding`	`kPdiPAV_ENCODING_BASE64`	The vObject uses Base 64 encoding. The writer outputs "`ENCODING=BASE64`."
`kPdiBEncoding`	`kPdiPAV_ENCODING_B`	The vObject uses Base 64 encoding. This is the same as the `kPdiB64Encoding` value, except that the PDI writer outputs "`ENCODING=B`."
		This encoding is used with the vCard 3.0 standard.
`kPdiEscapeEncoding`	`0x8000`	The vObject uses escapes for special characters.
`kPdiNoEncoding`	`0x8001`	The PDI writer does not encode the vObject value.

Error Code Constants

The PDI library functions return the error code constants shown in the following table to indicate their status.

Constant	Description
`pdiErrRead`	An error occurred while reading from the input stream.
`pdiErrWrite`	An error occurred while writing to the output stream.
`pdiErrNoPropertyName`	An attempt was made to write a property value before the property name was written.
`pdiErrNoPropertyValue`	The application did not write the last property value.
`pdiErrMoreChars`	The buffer is full. Superfluous characters have been discarded.
`pdiErrNoMoreFields`	There are no more property fields to read.
`pdiErrOpenFailed`	The PDI library could not be opened.
`pdiErrCloseFailed`	The PDI library could not be closed. This can occur if another application is using the library.

Parameter Name Constants

The `PdiConst.h` file defines several parameter name constants that you can use to specify the name of a parameter in functions that use parameter names. The parameter name constants have the following format:

`kPdiPAN_`*`parameterName`*

where *parameterValue* is replaced by a parameter name.

The following table shows examples of parameter name constants. For a complete list, see the `PdiConst.h` file.

Constant	Description
`kPdiPAN_TYPE`	The `TYPE` parameter.
`kPdiPAN_ENCODING`	The `ENCODING` parameter.
`kPdiPAN_STATUS`	The `STATUS` parameter.

Parameter Value Constants

The `PdiConst.h` file defines several parameter value constants that you can use to specify the name and value of a parameter in functions that use name and value pairs. The parameter value constants have the following format:

`kPdiPAV_parameterName_`*`parameterValue`*

where *parameterName* is replaced by a parameter name and *parameterValue* is replaced by a parameter value.

The following table shows examples of parameter value constants. For a complete list, see the `PdiConst.h` file.

Constant	Description
`kPdiPAV_TYPE_TEL`	The parameter name is `TYPE` and the parameter value is `TEL`.
`kPdiPAV_ENCODING_BASE64`	The parameter name is `ENCODING` and parameter value is `BASE64`.
`kPdiPAV_ENCODING_8BIT`	The parameter name is `ENCODING` and the parameter value is `8BIT`.
`kPdiPAV_STATUS_ACCEPTED`	The parameter name is `STATUS` and the parameter value is `ACCEPTED`.

Property Name Constants

The `PdiConst.h` file defines several property name constants that you can use to specify the name of a PDI property in functions that use property names. The property name constants have the following format:

`kPdiPRN_`*`propertyName`*

where *propertyName* is replaced by a property name.

The following table shows examples of property name constants. For a complete list, see the `PdiConst.h` file.

Constant	Description
`kPdiPRN_ADR`	The `ADR` property.
`kPdiPRN_BDAY`	The `BDAY` property.
`kPdiPRN_BEGIN`	The `BEGIN` property.
`kPdiPRN_BEGIN_VCARD`	The `BEGIN:VCARD` property.

Property Type Constants

You use the property type constants to specify the data type of a property.

Constant	Value	Description
`kPdiType_URI`	0	The data is a uniform resource identifier.
`kPdiType_UTC_OFFSET`	1	The data is an offset from UTC to local time.
`kPdiType_RECUR`	2	The data is a specification of a recurrence rule.
`kPdiType_PERIOD`	3	The data is a precise period of time.
`kPdiType_CAL_ADDRESS`	4	Calendar user data.
`kPdiType_BINARY`	5	Binary data.
`kPdiType_TEXT`	6	Text data.

Constant	Value	Description
`kPdiType_FLOAT`	`7`	Floating-point data.
`kPdiType_DURATION`	`8`	Time duration data.
`kPdiType_DATE_TIME`	`9`	Calendar date and time data.
`kPdiType_DATE`	`10`	Date data.
`kPdiType_INTEGER`	`11`	Signed integer data.
`kPdiType_TIME`	`12`	Time-of-day data.

Property Value Field Constants

The `PdiConst.h` file defines several property value field constants that you can use to specify the position of a PDI property value field in functions that use fields. The property value field constants have the following format:

`kPdiPVF_`*`propertyValueField`*

where *propertyValueField* is replaced by a property value field name.

The following table shows examples of property name constants. For a complete list, see the `PdiConst.h` file.

Constant	Description
`kPdiPVF_ADR_POST_OFFICE`	The property value field that stores the post office portion of the address.
`kPdiPVF_ADR_EXTENDED`	The property value field that stores the extended portion of the address.
`kPdiPAN_ADR_COUNTRY`	The property value field that stores the country portion of the address.

Property Value Format Constants

Some properties have **structured values**, which are values that contain multiple fields. These fields are typically separated by commas or semicolons in the vObject input or output stream. You use the property value format constants with the `PdiReadPropertyField()` and `PdiWritePropertyStr()` functions to specify how to handle fields in a structured value.

Constant	Value	Description
`kPdiNoFields`	`0`	There are no fields in the property value; `PdiReadPropertyField()` reads the entire value, or `PdiWritePropertyStr()` specifies that the entire value should be written.
`kPdiCommaFields`	`1`	Fields are separated with comma (“`,`“) characters; `PdiReadPropertyField()` reads one field, or `PdiWritePropertyStr()` specifies that one field should be written.
`kPdiSemicolonFields`	`2`	Fields are separated with semicolon (“`;`“) characters; `PdiReadPropertyField()` reads one field, or `PdiWritePropertyStr()` specifies that one field should be written.
`kPdiDefaultFields`	`4`	The parser decides the property value format, based on the property name.

Constant	Value	Description
`kPdiConvertComma`	`8`	Fields are separated with comma characters; `PdiReadPropertyField()` reads the entire value and converts each comma into a newline ("`\n`") character.
`kPdiConvertSemicolon`	`16`	Fields are separated with semicolon characters; `PdiReadPropertyField()` reads the entire value and converts each semicolon into a newline ("`\n`") character.

Reader and Writer Options Constants

You use the reader and writer option constants to control how the PDI reader (parser) reads values from the input stream or to control how the PDI writer (generator) writes values to the output stream.

Constant	Value	Description
`kPdiEnableFolding`	`1`	Enables folding of properties in the output stream. **Folding** is a mechanism for breaking long lines to allow them to be transmitted without change. If you specify this flag, the PDI library folds long lines. Note that folding is not compatible with earlier versions of the Palm OS®. Also note that other encoding formats, including quoted-printable and Base 64, define their own mechanisms for splitting long lines.
`kPdiEnableQuotedPrintable`	`2`	Enables quoted-printable encoding in the output stream and makes it the default encoding. This is an encoding format for non-ASCII values. You must have this enabled for compatibility with earlier versions of the Palm OS. If you do not specify this option, the default encoding is Base 64.

Constant	Value	Description
`kPdiEscapeMultiFieldValues`	`4`	For compatibility with earlier versions of the Palm OS. You must enable this for compatibility with earlier versions of the Palm OS. However, some non-Palm PDI software does not support this format. For more information about compatibility with earlier versions of the Palm OS, see Format Compatibility in the *Palm OS Programmer's Companion*, vol. II, *Communications*.
`kPdiPalmCompatibility`	`6`	This is a combination of `kPdiEnableQuotedPrintable | kPdiEscapeMultiFieldValues`. It forces the PDI writer to generate output that is compatible with earlier versions of the Palm OS.
`kPdiEnableB`	`8`	Enables base 64 encoding in the output stream, and tells the PDI writer to output "`ENCODING=B`" instead of "`ENCODING=BASE64`" when encoding a value with Base 64. Note: the vCard 3.0 standard has replaced the earlier `ENCODING=BASE64` with `ENCODING=B`. The meaning is the same.
`kPdiOpenParser`	`8`	Specifies that the PDI reader is open to all formats, including Palm and others.

Reader Event Constants

The PDI reader event constants specify the events that the reader has handled during the current read operation. The event values are combined together and stored in the `events` field of the PDI reader object. You can use them to test whether the reader handled a certain event.

Constant	Value	Description
`kPdiEOFEventMask`	1	End of file was reached.
`kPdiGroupNameEventMask`	2	A group name was found.
`kPdiPropertyNameEventMask`	4	A property name was found.
`kPdiParameterNameEventMask`	8	A parameter name was found.
`kPdiParameterValueEventMask`	16	A parameter value was found.
`kPdiPropertyDefinedEventMask`	32	A property definition was found; this implies that the ":" separator character was found.
`kPdiPropertyValueEventMask`	64	An entire property value was found
`kPdiPropertyValueFieldEventMask`	128	A value field was found; this implies that the ";" separator character was found.
`kPdiPropertyValueItemEventMask`	256	A value item was found; this implies that the "," separator character was found.

Constant	Value	Description
`kPdiPropertyValueMoreCharsEventMask`	`512`	The buffer that you provided was not large enough. The superfluous characters have been discarded.
`kPdiBeginObjectEventMask`	`1024`	The object begin indicator `BEGIN` was reached.
`kPdiEndObjectEventMask`	`2048`	The object end indicator `END` was reached.

Value Type Constants

You can use the following constants to specify data typing information for the `PdiWritePropertyBinaryValue()`, `PdiWritePropertyFields()`, and `PdiWritePropertyValue()` functions.

Constant	Value	Description
`kPdiWriteData`	`0`	The value is data. The PDI writer does not compute a character set. You can use this for binary data or pure ASCII data.
`kPdiWriteText`	`8`	The value is text data. The PDI writer parses the data character by character to compute the correct charset and character encoding for the data.
`kPdiWriteMultiline`	`16`	Explicitly specifies that the value contains special characters, such as newlines, and must be encoded. If this flag is not specified, the encoding is determined by the applied character set.

PDI Library Functions

PdiDefineReaderDictionary Function

Purpose Installs a new custom dictionary for use with a PDI reader object.

Declared In `PdiLib.h`

Prototype

```
PdiDictionary *PdiDefineReaderDictionary
    (UInt16 libRefnum, PdiReaderType *ioReader,
    PdiDictionary *dictionary,
    Boolean disableMainDictionary)
```

Parameters

→ *libRefnum*
The PDI library reference number.

→ *ioReader*
The PDI reader object with which to associate the dictionary. This object must have previously been created by a call to the `PdiReaderNew()` function.

→ *dictionary*
A pointer to a dictionary object that was created by the . The dictionary object is an array of binary data.

→ *disableMainDictionary*
If `true`, the main reader dictionary is disabled, and only this new dictionary is searched for terms; if `false`, the new dictionary supplements the main dictionary.

Returns Returns a pointer to the previously installed custom dictionary, or `NULL` if there was not a previously installed custom dictionary.

Comments This function installs a dictionary for use with the `ioReader` object. The dictionary contains the syntax for extensions or replacements of the PDI properties about which the PDI reader knows. The reader knows about properties specified in one of the vObject standards, including the vCard or vCal standards.

You can uninstall the current custom dictionary by specifying `NULL` as the value of the `dictionary` parameter,

Note that the dictionary must have previously been compiled by the dictionary tool. For more information, review the PDI sample code, which you can find on the Palm Developer's Knowledge Base at http://www.palmos.com/dev/tech/kb.

PdiDefineResizing Function

Purpose Defines the sizing information to use when automatically resizing a buffer. PDI reader objects read information from the input stream into a buffer and automatically adjust the buffer size as required.

Declared In `PdiLib.h`

Prototype

```
Err PdiDefineResizing(UInt16 libRefnum,
   PdiReaderType *ioReader, UInt16 deltaSize,
   UInt16 maxSize)
```

Parameters

→ *libRefnum*
The PDI library reference number.

→ *ioReader*
The PDI reader object, which was created by a previous call to the `PdiReaderNew()` function.

→ *deltaSize*
The number of bytes by which to grow the buffer when it needs resizing.

→ *maxSize*
The maximum allowable size for the buffer.

Returns Returns `errNone` if successful, and an error code if not successful.

Comments This function redefines the values to use when resizing a buffer. It does not perform any other actions.

The resizing values are used if your reader runs out of buffer space when storing input data during the processing of a property value. If possible, the reader resizes its internal buffer, using the values that you supply in this function.

The default resizing values apply if you do not call this function. The default values are:

```
kPdiDefaultBufferDeltaSize    0x0010
kPdiDefaultBufferMaxSize      0x3FFF
```

Compatibility Implemented only if 4.0 New Feature Set is present.

PdiDefineWriterDictionary Function

Purpose Installs a new custom dictionary for use with a PDI writer object.

Declared In `PdiLib.h`

Prototype

```
PdiDictionary *PdiDefineWriterDictionary
    (UInt16 libRefnum, PdiWriterType *ioWriter,
    PdiDictionary *dictionary,
    Boolean disableMainDictionary)
```

Parameters

→ *libRefnum*
The PDI library reference number.

→ *ioWriter*
The PDI writer object with which to associate the dictionary. This object must have previously been created by a call to the `PdiWriterNew()` function.

→ *dictionary*
A pointer to a dictionary object that was created by the . The dictionary object is an array of binary data.

→ *disableMainDictionary*
If `true`, the main dictionary is disabled, and only this new dictionary is searched for terms; if `false`, the new dictionary supplements the main dictionary.

Returns Returns a pointer to the previously installed custom dictionary, or `NULL` if there was not a previously installed custom dictionary.

Comments This function installs a dictionary for use with the `ioWriter` object. The dictionary contains the syntax for extensions or replacements of the PDI properties about which the PDI writer knows. The writer knows about properties specified in one of the vObject standards, including the vCard or vCal standards.

You can uninstall the current custom dictionary by specifying `NULL` as the value of the `dictionary` parameter,

Note that the dictionary must have previously been compiled by the dictionary tool. For more information, review the PDI sample code, which you can find on the Palm Developer's Knowledge Base at http://www.palmos.com/dev/tech/kb/.

PdiEnterObject Function

Purpose Tells the PDI library to enter into a recursively-defined object.

Declared In `PdiLib.h`

Prototype

```
Err PdiEnterObject(UInt16 libRefnum,
   PdiReaderType *ioReader)
```

Parameters

→ *`libRefnum`*
The PDI library reference number.

→ *`ioReader`*
The PDI reader object, which was created by a previous call to the `PdiReaderNew()` function.

Returns Returns `errNone` if successful, and an error code if not successful.

Comments Some vObjects recursively define other vObjects. Your application can choose whether or not to enter and parse the recursively defined objects.

If you want to parse the nested object definition, you need to call this function; otherwise, all of the properties of the nested object are skipped when the next call is made to the `PdiReadProperty()` or `PdiReadPropertyName()` functions.

Call this function after a `BEGIN_VObject` statement of the nested object has been parsed.

Compatibility Implemented only if 4.0 New Feature Set is present.

PdiLibClose Function

Purpose Close the PDI library after your application has finished with it.

Declared In `PdiLib.h`

Prototype

```
Err PdiLibClose(UInt16 libRefnum)
```

Parameters

→ *`libRefnum`*
The PDI library reference number.

Returns Returns 0 if no other application uses the library. You may need to call `SysLibRemove()` to remove the PDI library from the system library table.

Returns `pdiErrCloseFailed` if the library could not be closed.

Comments You should call this function after your application has finished with the PDI library, to allow the resources to be recovered.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiLibOpen()`

PdiLibOpen Function

Purpose Opens the PDI library for use by your application.

Declared In `PdiLib.h`

Prototype

```
Err PdiLibOpen(UInt16 libRefnum)
```

Parameters

→ *libRefnum*
The PDI library reference number.

Returns Returns `errNone` if successful, and an error code if not successful.

Comments You must call this function before calling any of the other PDI functions.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiLibClose()`

PdiParameterPairTest Macro

Purpose Determines if the reader has already parsed the specified parameter value or name-value pair.

Declared In `PdiLib.h`

Prototype

```
#define PdiParameterPairTest (reader, pair)
```

Parameters

→ *reader*
The PDI reader object, which was created by a previous call to the `PdiReaderNew()` function.

→ *pair*
The ID of the parameter. This must be one of the Parameter Value Constants.

Returns Returns `true` if the specified parameter name-value pair has been parsed for the current property, and `false` if not.

Comments Some vObject generators do not specify the parameter name if the name is considered evident from the context. This means that both of the following constructs are considered proper:

`Name=Value`

`Value`

The `PdiParameterPairTest` macro returns `true` if the value has been parsed in either format. For example,

```
PdiParameterPairTest(reader, kPdiPAV_TYPE_WORK)
```

returns true for either of the following:

`Type=WORK`

`WORK`

Compatibility Implemented only if 4.0 New Feature Set is present.

PdiReaderDelete Function

Purpose Delete a PDI reader object that is associated with the specified library number.

Declared In `PdiLib.h`

Prototype

```
void PdiReaderDelete(UInt16 libRefnum,
   PdiReaderType **ioReader)
```

Parameters

→ *libRefnum*
The PDI library reference number.

↔ *ioReader*
A pointer to the PDI reader object, which was created by a previous call to the `PdiReaderNew()` function.

Returns Returns nothing.

Comments This function deletes the `UDAReader` object associated with the reader object and frees the memory that was allocated for the reader object. The `ioReader` parameter is set to `NULL`.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiReaderNew()`

PdiReaderNew Function

Purpose Create and initialize a new PDI reader object for use with the specified PDI library number.

Declared In `PdiLib.h`

Prototype

```
PdiReaderType *PdiReaderNew(UInt16 libRefnum,
    UDAReaderType *input, UInt16 version)
```

Parameters

→ *`libRefnum`*
The PDI library reference number.

→ *`input`*
The Unified Data Access (UDA) input object associated with the reader.

→ *`version`*
Options to control the parsing behavior of the reader. You can use a combination of the Reader and Writer Options Constants.

Returns Returns a pointer to the new PDI reader object. Returns `NULL` if the reader cannot be created.

Comments The current implementation of the `PdiReaderNew` function does not make use of the `optionFlags` settings because the reader knows how to adapt itself to all of the supported formats. The options will be used in future versions.

The `input` value is a UDA object for reading data from an input stream that can be connected to various data sources. For example, you can use a `UDAExchangeReader` to read data from the Exchange Manager, and you can use a `UDAStringReader` to read data from a string. For more information about the UDA Manager, see Chapter 89, "Unified Data Access Manager."

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiReaderDelete()`, `PdiWriterNew()`

PdiReadParameter Function

Purpose Read a single parameter name and its value from an input stream.

Declared In `PdiLib.h`

Prototype

```
Err PdiReadParameter(UInt16 libRefnum,
   PdiReaderType *ioReader)
```

Parameters

→ *`libRefnum`*
The PDI library reference number.

→ *`ioReader`*
The PDI reader object, which was created by a previous call to the `PdiReaderNew()` function.

Returns

`0`
The parameter and its value were read successfully.

`kPdiReadError`
The parameter and its value could not be successfully read from the input stream.

This function initializes the `parameterName` and `parameter` fields of the `ioReader` object.

This function sets the appropriate bits in the reader's `parameterValues` field if the parameter name is recognized.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiReaderNew()`, `PdiReadPropertyField()`, `PdiReadPropertyName()`

PdiReadProperty Function

Purpose Read the next property and its parameters from the input stream.

Declared In `PdiLib.h`

Prototype

```
Err PdiReadProperty(UInt16 libRefnum,
   PdiReaderType *ioReader)
```

Parameters

→ *`libRefnum`*
The PDI library reference number.

→ *`ioReader`*
The PDI reader object, which was created by a previous call to the `PdiReaderNew()` function.

Returns Returns `errNone` if successful. Returns `kPdiReadError` if an error occurs.

Comments The `PdiReadProperty` function reads a property name and its parameters, by reading until it encounters the PDI "`:`" separator character.

This function looks each name up in the properties dictionary, and sets the appropriate bit in the `ioReader` object structure to indicate that property-parameter pair has been read. The properties dictionary stores information about properties that are considered well known, as described in The PDI Library Properties Dictionary in Chapter 3, "Personal Data Interchange," in *Palm OS Programmer's Companion*, vol. II, *Communications*.

To read a property, you call `PdiReadProperty`, followed by a call or calls to the `PdiReadPropertyField()` function to read the property value. For more information, see Reading Properties in Chapter 3, "Personal Data Interchange," in *Palm OS Programmer's Companion*, vol. II, *Communications*.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiReaderNew()`, `PdiReadPropertyField()`, `PdiReadPropertyName()`, `PdiReadParameter()`

PdiReadPropertyField Function

Purpose Read one field of a property value. The property value can be structured to contain multiple fields that are separated by commas or semicolons.

Declared In `PdiLib.h`

Prototype

```
Err PdiReadPropertyField(UInt16 libRefnum,
   PdiReaderType *ioReader, Char **bufferPP,
   UInt16 bufferSize, UInt16 readMode)
```

Parameters

→ *`libRefnum`*
The PDI library reference number.

→ *`ioReader`*
The PDI reader object, which was created by a previous call to the `PdiReaderNew()` function.

↔ *bufferPP*

A pointer to a pointer to the buffer into which the field characters are stored. Set this value to `NULL` to allow the PDI library to manage it.

Note that the PDI library may need to resize the buffer; thus, the value of this parameter might change.

→ *bufferSize*

The size, in bytes, of the input buffer for reading the field.

You can use the `PdiResizableBuffer` constant to specify that the PDI Library can automatically resize the buffer as required.

If you do not specify the `PdiResizableBuffer` value, then the PDI library assumes that `buffer` cannot be moved, and that its size is fixed.

→ *readMode*

The format of the fields in the property value. Use one of the Property Value Format Constants.

Returns

`0`

The field was read successfully.

`kPdiNoMoreFieldsError`

There are no more fields to read because the entire value has already been read.

`kPdiMoreCharsError`

The buffer is not large enough to store the entire field.

Comments

The value returned in the `buffer` is terminated with the "`\0`" character.

If the field is an empty string, the buffer is erased from memory, and the value of `buffer` is set to `NULL`.

If you specify `kPdiResizableBuffer` for the value of the `bufferSize` parameter, and the buffer needs more space, `PdiReadPropertyField` resizes the buffer for you, which may cause the value of `buffer` to be modified.

This function initializes the `propertyValue` and `fieldNum` fields of the `ioReader` object.

To read a property, you usually call the `PdiReadProperty()` function, followed by a call or calls to `PdiReadPropertyField` to

read the property value. For more information, see Reading Properties in Chapter 3, "Personal Data Interchange," in *Palm OS Programmer's Companion*, vol. II, *Communications*.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiReaderNew()`, `PdiReadProperty()`, `PdiReadPropertyName()`, `PdiReadParameter()`

PdiReadPropertyName Function

Purpose Read a property name from an input stream. Use this function when you want to parse and process each parameter individually.

Declared In `PdiLib.h`

Prototype

```
Err PdiReadPropertyName(UInt16 libRefnum,
   PdiReaderType *ioReader)
```

Parameters

→ *`libRefnum`*
The PDI library reference number.

→ *`ioReader`*
The PDI reader object, which was created by a previous call to the `PdiReaderNew()` function.

Returns Returns `errNone` if successful, and an error code if not successful.

Comments The `PdiReadProperty` function reads a property name only, reading until it encounters the PDI "`:`" separator character, or until it encounters the first parameter "`,`" separator character.

This function initializes the `property` and `propertyName` fields of the `ioReader` object.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiReaderNew()`, `PdiReadPropertyField()`, `PdiReadProperty()`, `PdiReadParameter()`

PdiSetCharset Function

Purpose Force the character set of the next property value that is written by the specified PDI writer.

Declared In `PdiLib.h`

Prototype

```
Err PdiSetCharset(UInt16 libRefnum,
    PdiWriterType *ioWriter,
    CharEncodingType charset)
```

Parameters

→ *libRefnum*
The PDI library reference number.

→ *ioWriter*
The PDI writer object, which was created by a previous call to the `PdiWriterNew()` function.

→ *charset*
The character set to use for the property value. This must be one of the following `CharEncodingType` values:

`charEncodingAscii`

`charEncodingISO8859_1`

`charEncodingShiftJIS`

`charEncodingISO2022Jp`

Returns Returns `errNone` if successful, and an error code if not successful.

Comments This function tells `ioWriter` to use the specified `charSet` for the next property value that it writes, rather than computing a character set for that value.

You can determine the current character setting by examining the `charset` field of your PDI writer object.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiSetEncoding()`

PdiSetEncoding Function

Purpose Force the encoding of the current property value.

Declared In `PdiLib.h`

Prototype

```
Err PdiSetEncoding(UInt16 libRefnum,
   PdiWriterType *ioWriter, UInt16 encoding)
```

Parameters

→ *libRefnum*
The PDI library reference number.

→ *ioReader*
The PDI writer object, which was created by a previous call to the `PdiWriterNew()` function.

→ *encoding*
The encoding to apply to the property value. This must be one of the Encoding Type Constants.

Returns Returns `errNone` if successful, and an error code if not successful.

Comments This function changes the encoding for the property value to the specified `encoding` value

You can determine the current encoding setting by examining the `encoding` field of your PDI writer object.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiSetCharset()`

PdiWriteBeginObject Function

Purpose Writes a vObject begin tag to an output stream.

Declared In `PdiLib.h`

Prototype

```
Err PdiWriteBeginObject(UInt16 libRefnum,
   PdiWriterType *ioWriter, UInt16 objectNameID)
```

Parameters

→ *libRefnum*
The PDI library reference number.

→ *ioWriter*
The PDI writer object, which was created by a previous call to the `PdiWriterNew()` function.

→ *objectNameID*
The object name ID. This must be one of the Property Name Constants that begins an object, including the following:

`kPdiPRN_BEGIN_VCAL`

`kPdiPRN_BEGIN_VCAL`

`kPdiPRN_BEGIN_VCARD`

`kPdiPRN_BEGIN_VEVENT`

`kPdiPRN_BEGIN_VFREEBUSY`

`kPdiPRN_BEGIN_VJOURNAL`

`kPdiPRN_BEGIN_VTIMEZONE`

`kPdiPRN_BEGIN_VTODO`

Returns Returns `errNone` if successful, and an error code if not successful.

Comments Call this function to begin writing a vObject to the output stream. It writes a begin tag such as "`BEGIN:VCARD`" to the output stream.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiWriteEndObject()`, `PdiWriteProperty()`

PdiWriteEndObject Function

Purpose Writes a vObject end tag to an output stream.

Declared In `PdiLib.h`

Prototype

```
Err PdiWriteEndObject(UInt16 libRefnum,
   PdiWriterType *ioWriter, UInt16 objectNameID)
```

Parameters

→ *libRefnum*
The PDI library reference number.

→ *ioWriter*
The PDI writer object, which was created by a previous call to the `PdiWriterNew()` function.

→ *objectNameID*
The object name ID. This must be one of the Property Name Constants that ends an object, including the following:

`kPdiPRN_END_VCAL`

```
kPdiPRN_END_VCAL
kPdiPRN_END_VCARD
kPdiPRN_END_VEVENT
kPdiPRN_END_VFREEBUSY
kPdiPRN_END_VJOURNAL
kPdiPRN_END_VTIMEZONE
kPdiPRN_END_VTODO
```

Returns Returns `errNone` if successful, and an error code if not successful.

Comments Call this function to finish writing a vObject to the output stream. It writes a end tag such as "`END:VCARD`" to the output stream.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiWriteBeginObject()`, `PdiWriteProperty()`

PdiWriteParameter Function

Purpose Write a parameter, and optionally its name, to an output stream.

Declared In `PdiLib.h`

Prototype

```
Err PdiWriteParameter(UInt16 libRefnum,
   PdiWriterType *ioWriter, UInt16 parameter,
   Boolean parameterName)
```

Parameters

→ *libRefnum*
The PDI library reference number.

→ *ioWriter*
The PDI writer object, which was created by a previous call to the `PdiWriterNew()` function.

→ *parameter*
The ID of the parameter. This must be one of the Parameter Value Constants.

→ *parameterName*

If this is `true`, the parameter name, followed by the "=" symbol, followed by the parameter value is written to the output stream.

If this is `false`, only the parameter value is written to the output stream.

Returns Returns `errNone` if successful, and an error code if not successful.

Comments Use this function to write a parameter to the output stream. To write a property, you usually call the `PdiWriteProperty()` function, followed by calls to `PdiWriteParameter` to write any parameters, followed by a call to the `PdiWritePropertyValue()` function to write the property value. For more information, see Writing Properties in Chapter 3, "Personal Data Interchange," in *Palm OS Programmer's Companion*, vol. II, *Communications*.

You can use the `parameterName` argument to specify that you want the parameter name written as well as the parameter value. For example, the following table shows what is written if the value of `parameter` is `kPdiPAV_TYPE_HOME`.

Value of parameterName	Data written to output stream
`true`	`TYPE=HOME`
`false`	`HOME`

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiWriteProperty()`, `PdiWritePropertyValue()`, `PdiWritePropertyFields()`, `PdiWritePropertyStr()`, `PdiWriteParameterStr()`

PdiWriteParameterStr Function

Purpose Write a parameter name and the parameter value to an output stream.

Declared In `PdiLib.h`

Prototype

```
Err PdiWriteParameterStr(UInt16 libRefnum,
    PdiWriterType *ioWriter,
    const Char *parameterName,
    const Char *parameterValue)
```

Parameters

→ *libRefnum*
The PDI library reference number.

→ *ioWriter*
The PDI writer object, which was created by a previous call to the `PdiWriterNew()` function.

→ *parameterName*
The name of the parameter. If the value of this is the empty string or `NULL`, only the parameter value is written.

→ *parameterValue*
The parameter value string.

Returns Returns `errNone` if successful, and an error code if not successful.

Comments This function writes the parameter name, followed by the "=" symbol, followed by the parameter value, to the output stream. If `parameterName` is `NULL`, or if its value is the empty string, just the parameter value is written.

This function is similar to the `PdiWriteParameter()` function. The difference is that `PdiWriteParameterStr` takes the name and value of the parameter as strings, while `PdiWriteParameter` takes them as ID constants.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiWriteProperty()`, `PdiWritePropertyValue()`, `PdiWritePropertyFields()`, `PdiWritePropertyStr()`, `PdiWriteParameter()`

PdiWriteProperty Function

Purpose Writes a property name to an output stream.

Declared In `PdiLib.h`

Prototype

```
Err PdiWriteProperty(UInt16 libRefnum,
    PdiWriterType *ioWriter,
    UInt16 propertyNameID)
```

Parameters

→ *`libRefnum`*
The PDI library reference number.

→ *`ioWriter`*
The PDI writer object, which was created by a previous call to the `PdiWriterNew()` function.

→ *`propertyNameID`*
The property name to write. This must be one of the Property Name Constants.

Returns Returns `errNone` if successful, and an error code if not successful.

Comments To write a property, you usually call `PdiWriteProperty`, followed by calls to the `PdiWriteParameter()` function to write any parameters, followed by a call to the `PdiWritePropertyValue()` function to write the property value. For more information, see Writing Properties in Chapter 3, "Personal Data Interchange," in *Palm OS Programmer's Companion*, vol. II, *Communications*.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiWritePropertyValue()`, `PdiWritePropertyFields()`, `PdiWritePropertyStr()`, `PdiWriteParameter()`

PdiWritePropertyBinaryValue Function

Purpose Write a binary property value to an output stream.

Declared In `PdiLib.h`

Prototype

```
Err PdiWritePropertyBinaryValue(UInt16 libRefnum,
    PdiWriterType *ioWriter, const Char *buffer,
    UInt16 size, UInt16 options)
```

Parameters

→ *`libRefnum`*
The PDI library reference number.

→ *ioWriter*
The PDI writer object, which was created by a previous call to the `PdiWriterNew()` function.

→ *buffer*
A buffer that contains the binary data.

→ *size*
The number of bytes of data to write from the buffer.

→ *options*
The data type. This must be a combination of one or more of the Value Type Constants.

Returns Returns `errNone` if successful, and an error code if not successful.

Comments Use this function to write a binary data property value, such as a sound or an image.

This function encodes the data when it is written. The character set that gets applied to the data is not computed by this function; however, you can call the `PdiSetCharset()` function to set the character set.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiWriteProperty()`, `PdiWritePropertyFields()`, `PdiWritePropertyValue()`

PdiWritePropertyFields Function

Purpose Write a structured property value with multiple fields to an output stream.

Declared In `PdiLib.h`

Prototype

```
Err PdiWritePropertyFields(UInt16 libRefnum,
   PdiWriterType *ioWriter, Char *fields[],
   UInt16 fieldNumber, UInt16 options)
```

Parameters

→ *libRefnum*
The PDI library reference number.

→ *ioWriter*
The PDI writer object, which was created by a previous call to the `PdiWriterNew()` function.

→ *fields*
An array of strings, each of which is a field of the property value. Individual fields can be `NULL`.

→ *fieldNumber*
The number of fields in the `Fields` array.

→ *options*
The data type. This must be a combination of one or more of the Value Type Constants.

Returns Returns `errNone` if successful, and an error code if not successful.

Comments Use this function to write a property value that contains multiple fields.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiWritePropertyValue()`, `PdiWritePropertyBinaryValue()`, `PdiReadPropertyField()`

PdiWritePropertyStr Function

Purpose Writes the name of a property to the output stream, and specifies the property value's structure for subsequent write operations.

Declared In `PdiLib.h`

Prototype

```
Err PdiWritePropertyStr(UInt16 libRefnum,
    PdiWriterType *ioWriter,
    const Char *propertyName, UInt8 writeMode,
    UInt8 requiredFields)
```

Parameters

→ *libRefnum*
The PDI library reference number.

→ *ioWriter*
The PDI writer object, which was created by a previous call to the `PdiWriterNew()` function.

→ *propertyName*
The name of the property to write.

→ *writeMode*
The format of the fields in the property value. Use one of the following Property Value Format Constants:

`kPdiNoFields`

`kPdiCommaFields`

`kPdiSemicolonFields`

→ *`requiredFields`*
The number of required fields for the value. This is usually set to 1.

Returns Returns `errNone` if successful, and an error code if not successful.

Comments Use this function when you are writing a property that is not in the dictionary, or when you are writing a property that uses value formatting that differs from the standard formatting stored in the dictionary for the property name.

This function writes the property name to the output stream, and then establishes the structure of the property's value, including the number of required fields and the separator character for those fields. After calling this function, the next call to the `PdiWritePropertyValue()` or `PdiWritePropertyFields()` functions correctly writes the property value.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiWriteProperty()`, `PdiWritePropertyValue()`, `PdiWritePropertyFields()`, `PdiWriteParameter()`

PdiWritePropertyValue Function

Purpose Write a string to the output stream as the entire value of a property.

Declared In `PdiLib.h`

Prototype

```
Err PdiWritePropertyValue(UInt16 libRefnum,
   PdiWriterType *ioWriter, Char *buffer,
   UInt16 options)
```

Parameters

→ *`libRefnum`*
The PDI library reference number.

→ *`ioWriter`*
The PDI writer object, which was created by a previous call to the `PdiWriterNew()` function.

→ *buffer*
The input buffer that contains the string to be written.

→ *options*
The data type. This must be a combination of one or more of the Value Type Constants.

Returns Returns `errNone` if successful, and an error code if not successful.

Comments Use this function to write a property value that contains a single field.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiWriteProperty()`, `PdiWritePropertyFields()`, `PdiWriteParameter()`, `PdiWritePropertyBinaryValue()`

PdiWriterDelete Function

Purpose Delete a PDI output stream object.

Declared In `PdiLib.h`

Prototype

```
void PdiWriterDelete(UInt16 libRefnum,
   PdiWriterType **ioWriter)
```

Parameters

→ *libRefnum*
The PDI library reference number.

↔ *ioWriter*
A pointer to the PDI writer object, which was created by a previous call to the `PdiWriterNew()` function.

Returns Returns nothing.

Comments This function frees the memory that was allocated for the writer object. The `ioWriter` parameter is set to `NULL`.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiWriterNew()`

PdiWriterNew Function

Purpose Initializes a new PDI writer object for use with the specified library number.

Declared In `PdiLib.h`

Prototype

```
PdiWriterType *PdiWriterNew(UInt16 libRefnum,
   UDAWriterType *output, UInt16 version)
```

Parameters

→ *`libRefnum`*
The PDI library reference number.

→ *`output`*
The Unified Data Access (UDA) output object associated with the writer.

→ *`version`*
Options to control the behavior of the writer. You can use a combination of the Reader and Writer Options Constants.

Returns Returns a pointer to the new PDI writer object. Returns `NULL` if the reader cannot be created.

Comments The `media` pointer is copied into a field in the writer object; thus, you do not need to retain your copy.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `PdiWriterDelete()`, `PdiReaderNew()`

89

Unified Data Access Manager

This chapter provides reference material for the Unified Data Access (UDA) Manager, which provides a mechanism for abstracting read and write access to different kinds of source and destination media, including memory and the Exchange Manager.

The Personal Data Interchange (PDI) reader and writer objects use UDA objects, and you must create UDA objects to use the PDI functions.

This chapter discusses the following topics:

- UDA Manager Data Structures
- UDA Manager Constants
- UDA Manager Functions
- UDA Object Creation Functions

The header file `UDAMgr.h` declares the Unified Data Access Manager API.

You use the UDA Manager in conjunction with the PDI library. For more information about the PDI library, see Chapter 88, "Personal Data Interchange Library."

Chapter 3, "Personal Data Interchange," in the *Palm OS Programmer's Companion*, vol. II, *Communications*, provides examples of using the UDA functions with the PDI library.

UDA Manager Data Structures

UDABufferSize Typedef

Purpose The `UDABufferSize` type is a simple typedef that defines the size of buffers used with UDA read functions.

Prototype

```
typedef UInt16 UDABufferSize;
```

UDAObjectType Struct

Purpose The `UDAObjectType` is the base class for all UDA objects, and defines the common properties of all of the objects.

Prototype

```
typedef struct UDAObjectTag {
   UInt16 optionFlags;
   UDADeleteFunction deleteF;
   UDAControlFunction controlF;
} UDAObjectType;
```

Fields

`optionFlags`
: Options for the object. This is a combination of values described in Object Option Flags.

`deleteF`
: The delete function associated with this UDA object.

`controlF`
: The control function associated with this UDA object.

UDAFilterType Struct

Purpose The `UDAFilterType` represents UDA Filters, which take input from a UDA Reader or UDA Filter, perform some encoding or decoding operation, and output the data to a memory buffer.

Prototype

```
typedef struct UDAFilterTag {
   UInt16 optionFlags;
   UDADeleteFunction deleteF;
   UDAControlFunction controlF;
   UDAReadFunction readF;
   UDAReaderType *upperReader;
} UDAFilterType;
```

Fields

`optionFlags`
: Options for the object. This is a combination of values described in Object Option Flags.

`deleteF`
: The delete function associated with this UDA object.

`controlF`
: The control function associated with this UDA object.

`readF`
: The read function associated with this UDA object.

`upperReader`
: The `UDAReaderType` or `UDAFilterType` object that reads the data that this object outputs.

UDAReaderType Struct

Purpose

The `UDAReaderType` represents UDA Readers, which read input from a medium.

Prototype

```
typedef struct UDAReaderTag {
   UInt16 optionFlags;
   UDADeleteFunction deleteF;
   UDAControlFunction controlF;
   UDAReadFunction readF;
} UDAReaderType;
```

Fields

`optionFlags`
: Options for the object. This is a combination of values described in Object Option Flags.

`deleteF`
: The delete function associated with this UDA object.

`controlF`
: The control function associated with this UDA object.

`readF`
: The read function associated with this UDA object.

UDAWriterType Struct

Purpose

The `UDAWriterType` represents UDA Writers, which take data from a UDA Reader or UDA Filter and write the data to an output medium.

Prototype

```
typedef struct UDAWriterTag {
   UInt16 optionFlags;
   UDADeleteFunction deleteF;
   UDAControlFunction controlF;
   UDAWriteFunction initiateWriteF;
   UDAFlushFunction flushF;
   UDAReaderType *upperReader;
} UDAWriterType;
```

Fields

`optionFlags`
: Options for the object. This is a combination of values described in Object Option Flags.

`deleteF`
: The delete function associated with this UDA object.

`controlF`
: The control function associated with this UDA object.

`initiateWriteF`
: The write function associated with this UDA object.

`flushF`
: The flush function associated with this UDA object.

`upperReader`
: The `UDAReaderType` object that reads the data that this object writes.

UDA Manager Constants

This section describes the constants used with the UDA Manager, which include the following constant types:

- Control Flags

- Error Constants
- Object Option Flags
- Miscellaneous Constants

Control Flags

Use the control flag constants to control UDA objects with the `UDAControl()` macro.

Constant	Value	Description
`kUDAReinitialize`	1	Used with the `UDAControl()` macro to reinitialize the UDA object.

Error Constants

At the time of this writing, there is only one error constant associated with the UDA object API.

Constant	Description
`udaErrControl`	Returned by the `UDAControl()` macro when the control parameter is not valid for the UDA object.

Object Option Flags

You use the object option flag constants to determine information about the internal state of UDA objects. Note that the `UDAEndOfReader()` and `UDAMoreData()` macros provide you with a convenient means of accessing this same information.

Constant	Value	Description
`kUDAEndOfReader`	1	Indicates that the UDA reader has reached the end of its data.
`kUDAMoreData`	2	Indicates that the UDA reader needs more space in the read buffer to do its work.

Miscellaneous Constants

Constant	Value	Description
`kUDAZeroTerminatedBuffer`	`0xFFFF`	Indicates that the buffer is a null-terminated string. Use this value when creating or reinitializing a `UDAMemoryReader` object.

UDA Manager Functions

UDAControl Function

Purpose Applies controls to a UDA object.

Declared In `UDAMgr.h`

Prototype

```
Err UDAControl (UDAObjectType *ioObjectP,
   UInt16 parameter, va_args)
```

Parameters

→ *`ioObjectP`*
A pointer to the `UDAObjectType` object that you want to control. This can be a `UDAReaderType`, a `UDAFilterType`, or a `UDAWriterType` object.

→ *`parameter`*
The control action that you want applied to the object.

→ *`va_args`*
Additional parameters, as required for the control and object type.

Returns Returns `errNone` if no error, or `udaErrorClass` if the control parameter is not valid for the `ioObjectP`.

Comments The `UDAControl` function applies a control action to a UDA object. You may need to supply additional parameters, depending on the object type and control parameter values.

The only control action defined in Palm OS 4.0 is `kUDAReinitialize`. You can use it as shown in Table 89.1.

Table 89.1 UDAControl actions

Object Type	Usage	Action
`UDAExchangeReaderType`	`UDAControl(myExgRdr, kUDAReinitialize)`	Does nothing
`UDAExchangeWriterType`	`UDAControl(myExgWtr, kUDAReinitialize)`	Does nothing
`UDAMemoryReaderType`	`UDAControl(myMemRdr, kUDAReinitialize, bufferP, bufferSize)`	Reinstalls a new buffer for the memory reader. See `UDAMemoryReaderNew()` for more information about the parameters.

Compatibility Implemented only if 4.0 New Feature Set is present.

UDADelete Macro

Purpose Deletes a UDA object.

Declared In `UDAMgr.h`

Prototype

```
#define UDADelete (ioObjectP)
```

Parameters → *`ioObjectP`*

A pointer to the `UDAObjectType` object that you want to delete. This can be a `UDAReaderType`, a `UDAFilterType`, or a `UDAWriterType` object.

Returns Returns nothing.

Comments The `ioObjectP` pointer is not valid after this macro completes.

Compatibility Implemented only if 4.0 New Feature Set is present.

UDAEndOfReader Macro

Purpose Tests if the end of the reader has been reached.

Declared In `UDAMgr.h`

Prototype

```
#define UDAEndOfReader (ioReaderP)
```

Parameters

→ *`ioReaderP`*
A pointer to a `UDAReaderType` object.

Returns Returns `true` if the end of the reader referenced by `ioReaderP` has been reached, and `false` if not.

Comments The end of the reader has been reached.

Compatibility Implemented only if 4.0 New Feature Set is present.

UDAFilterJoin Macro

Purpose Joins a filter with a reader.

Declared In `UDAMgr.h`

Prototype

```
#define UDAFilterJoin (ioFilterP, newReaderP)
```

Parameters

→ *`ioFilterP`*
A pointer to a `UDAFilterType` object.

→ *`newReaderP`*
A pointer to the `UDAReaderType` object with which you want the filter joined.

Returns Returns nothing.

Comments Each `UDAFilterType` object receives its data from the `UDAReaderType` object to which it is joined; this relationship is established when you create the filter object. You can use this macro to change the reader with which the filter is joined. Upon completion, the filter referenced by `ioFilterP` is joined with the reader referenced by `newReaderP`.

Compatibility Implemented only if 4.0 New Feature Set is present.

UDAInitiateWrite Macro

Purpose Causes the `UDAWriterType` object to read data and then write that data to output.

Declared In `UDAMgr.h`

Prototype

```
#define UDAInitiateWrite (ioWriterP)
```

Parameters → *ioWriterP*
A pointer to a `UDAWriterType` object.

Returns Returns `errNone` if successful, and an error code if not.

Comments When you use this macro, the `ioWriterP` reads data from the reader to which it is joined. It reads data until the reader is empty, and then writes the data to the output medium.

Compatibility Implemented only if 4.0 New Feature Set is present.

UDAMoreData Macro

Purpose Tests if there is more data available to read, but not enough room in the buffer to read it in.

Declared In `UDAMgr.h`

Prototype

```
#define UDAMoreData (ioReaderP)
```

Parameters → *ioReaderP*
A pointer to a `UDAReaderType` object.

Returns Returns `true` if there is more data available for the reader and `false` if there is no more data available.

Comments You can use this macro with `UDAReaderType` objects to determine if there is more data waiting to read. This can happen when the reader's buffer is full.

Compatibility Implemented only if 4.0 New Feature Set is present.

UDARead Macro

Purpose Uses the specified `UDAReaderType` object to read data from the input source and place that data into the specified buffer.

Declared In `UDAMgr.h`

Prototype

```
#define UDARead (ioReaderP, bufferToFillP,
    bufferSize, errorP)
```

Parameters

→ *`ioReaderP`*
A pointer to a `UDAReaderType` object that performs the read.

→ *`bufferToFillP`*
A pointer to the buffer into which data is placed.

→ *`bufferSize`*
The size of the buffer, in bytes.

→ *`errorP`*
A pointer to an `Err` value that represents the result of the read operation; if the operation is successful, the value is set to `errNone`.

Returns Returns the number of bytes that were actually read. This value can be less than or equal to the value of `bufferSize`.

Comments The reader reads from the input source associated with the reader object and places the data into the specified buffer, reading no more than `bufferSize` bytes of data.

Compatibility Implemented only if 4.0 New Feature Set is present.

UDAWriterFlush Macro

Purpose Flushes the buffer used by the `UDAWriterType` object.

Declared In `UDAMgr.h`

Prototype

```
#define UDAWriterFlush (ioWriterP)
```

Parameters

→ *`ioWriterP`*
A pointer to a `_UDAWriterType` object.

Returns Returns `errNone` if successful, and an error code if not.

Comments You can use this macro to flush any data remaining in the buffer of the writer object referenced by `ioWriterP`. This causes any data in the buffer to be sent to the output medium.

Compatibility Implemented only if 4.0 New Feature Set is present.

UDAWriterJoin Macro

Purpose Joins a writer object to a different reader object.

Declared In `UDAMgr.h`

Prototype

```
#define UDAWriterJoin (ioWriterP, newReaderP)
```

Parameters

→ *ioWriterP*
A pointer to a `UDAWriterType` object.

→ *newReaderP*
A pointer to the `UDAReaderType` object with which you want the writer joined.

Returns Returns nothing.

Comments Each `UDAWriterType` object receives its data from the `UDAReaderType` object to which it is joined; this relationship is established when you create the writer object. You can use this macro to change the reader with which the writer is joined. Upon completion, the writer referenced by `ioWriterP` is joined with the reader referenced by `newReaderP`.

Compatibility Implemented only if 4.0 Feature Set is present.

UDA Object Creation Functions

UDAExchangeReaderNew Function

Purpose Creates a new `UDAReaderType` object that you can use to read data from an Exchange Manager socket.

Declared In `UDAMgr.h`

Prototype

```
UDAReaderType *UDAExchangeReaderNew
    (ExgSocketPtr socket)
```

Parameters

→ *socket*
A pointer to an `ExgSocketType` structure that describes the connection.

Returns Returns a pointer to the newly created `UDAReaderType` object, or `NULL` if the reader could not be created.

Comments Use this function to create a UDA Reader object that takes input from an Exchange Manager socket.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `ExgSocketType`

UDAExchangeWriterNew Function

Purpose Creates a new `UDAWriterType` object that you can use to write data to an Exchange Manager socket.

Declared In `UDAMgr.h`

Prototype

```
UDAWriterType *UDAExchangeWriterNew
    (ExgSocketPtr socket,
    UDABufferSize bufferSize)
```

Parameters

→ *socket*
A pointer to an `ExgSocketType` structure that describes the connection.

→ *bufferSize*
The size, in bytes, of the buffer for the new writer object.

Returns Returns a pointer to the newly created UDA Writer, or NULL if the writer could not be created.

Comments Use this function to create a UDA Writer object that sends output to an Exchange Manager socket.

Compatibility Implemented only if 4.0 New Feature Set is present.

See Also `ExgSocketType`

UDAMemoryReaderNew Function

Purpose Creates a new `UDAReaderType` object that you can use to read data from a memory buffer.

Declared In `UDAMgr.h`

Prototype

```
UDAReaderType *UDAMemoryReaderNew
    (const UInt8 *bufferP,
    UDABufferSize bufferSizeInBytes)
```

Parameters

→ *bufferP*
A pointer to a buffer in memory from which data is read.

→ *bufferSize*
The size of the buffer, in bytes. If this value is equal to `kUDAZeroTerminatedBuffer`, `bufferP` must point to a null-terminated string buffer.

Returns Returns a pointer to the newly created `UDAReaderType` object, or `NULL` if the reader could not be created.

Comments Use this function to create a reader that takes input from memory.

Compatibility Implemented only if 4.0 New Feature Set is present.

Part V
Appendixes

A

System Use Only Functions

This appendix lists functions that are purposely undocumented because they are for system use only.

WARNING! System Use Only.

```
AbtShowAbout
AddrDBSort
AddrJDBSort
AlmAlarmCallback
AlmCancelAll
AlmDisplayAlarm
AlmEnableNotification
AlmInit
AlmTimeChange
BtLibHandleEvent
BtLibHandleTransportEvent
BtLibMutex
BtLibOpenBackground
BtLibServiceClose
BtLibServiceIndicateSessionStart
BtLibServiceOpen
BtLibServicePlaySound
BtLibSleep
BtLibUnload
BtLibWake
Crc16CalcBigBlock
DmInit
DmResetRecordStates
ErrAlertCustom
EvtDequeueKeyEvent
EvtEnqueuePenPoint
```

EvtGetSysEvent

EvtInitialize

EvtSetKeyQueuePtr

EvtSetPenQueuePtr

EvtSysInit

ExgInit

ExgNotifyReceiveV35

ExpCardInserted

ExpCardRemoved

ExpInit

ExpSlotRegister

ExpSlotUnregister

FileReadLow

Find

FrmActiveState

FrmAddSpaceForObject

FrmGetUserModifiedState

FrmSetNotUserModified

FtrInit

GrfBeginStroke

GrfFieldChange

GrfFree

GrfInit

HostControl

INetLibSleep

INetLibSockMailAttrGet

INetLibSockMailAttrSet

INetLibSockMailQueryProgress

INetLibSockMailReqAdd

INetLibSockMailReqCreate

INetLibSockMailReqSend

INetLibWake

InsPtCheckBlink

InsPtInitialize

IntlInit

IrHandleEvent

IrWaitForEvent

KeyboardStatusNew

KeyboardStatusFree

KbdSetLayout

KbdGetLayout

KbdSetPosition

KbdGetPosiiton

KbdSetShiftState

KbdGetShiftState

KbdDraw

KbdErase

KbdHandleEvent

MemCardFormat

MemChunkFree

MemChunkNew

MemHandleFlags

MemHandleLockCount

MemHandleOwner

MemHandleResetLock

MemHeapFreeByOwnerID

MemHeapInit

MemInit

MemInitHeapTable

MemKernelInit

MemPtrFlags

MemPtrOwner

MemPtrResetLock

MemSemaphoreRelease

MemSemaphoreReserve

MemStoreSetInfo

NetLibConfigAliasGet

NetLibConfigAliasSet

NetLibConfigDelete

NetLibConfigIndexFromName

NetLibConfigList

NetLibConfigMakeActive

NetLibConfigRename

NetLibConfigSaveAs

NetLibHandlePowerOff

NetLibOpenConfig

NetLibOpenIfCloseWait

NetLibSleep

NetLibWake

PenClose

PenGetRawPen

PenOpen

PenRawToScreen

PenScreenToRaw

PenSleep

PenWake

ResLoadString

ScrCompressScanLine

ScrCopyRectangle

ScrDeCompressScanLine

ScrDrawChars

ScrDrawNotify

ScrInit

ScrLineRoutine

ScrRectangleRoutine

ScrScreenInfo

ScrSendUpdateArea

SerDbgAssureOpen

SerialMgrInstall

SerReceiveISP

SerReceiveWindowClose

SerReceiveWindowOpen

SerSetMapPort

SerSetWakeupHandler

SerSleep

SerWake

SlkProcessRPC

SlkSysPktDefaultResponse

SndInit

SndSetDefaultVolume (in Palm OS Garnet; this function may be used with prior Palm OS versions)

SndStreamDeviceControl

SrmSelectorErrPrv

SrmSleep
SrmWake
SysAppStartup
SysAppExit
SysBatteryDialog
SysCardImageDeleted
SysCardImageInfo
SysColdBoot
SysDisableInts
SysDoze
SysEvGroupCreate
SysEvGroupRead
SysEvGroupSignal
SysEvGroupWait
SysInit
SysKernelInfo
SysLaunchConsole
SysLCDBrightness
SysLCDContrast
SysLibClose
SysLibInstall
SysLibOpen
SysLibSleep
SysLibTblEntry
SysLibWake
SysMailboxCreate
SysMailboxDelete
SysMailboxFlush

SysMailboxSend

SysMailboxWait

SysNewOwnerID

SysPowerOn

SysResSemaphoreCreate

SysResSemaphoreDelete

SysResSemaphoreRelease

SysResSemaphoreReserve

SysRestoreStatus

SysSemaphoreCreate

SysSemaphoreDelete

SysSemaphoreSet

SysSemaphoreSignal

SysSemaphoreWait

SysSetA5

SysSetPerformance

SysSleep

SysTaskCreate

SysTaskDelete

SysTaskID

SysTaskResume

SysTaskSetTermProc

SysTaskSuspend

SysTaskSwitching

SysTaskTrigger

SysTaskWait

SysTaskWaitClr

SysTaskWake

SysTimerCreate
SysTimerDelete
SysTimerRead
SysTimerWrite
SysTranslateKernelErr
SysUIBusy
SysUILaunch
SysUnimplemented
SysWantEvent
TimInit
TxtPrepFindString
UIColorPopTable
UIColorPushTable
UIInitialize
UIReset
UIPopTable
UIPushTable
VFSInit
WinAddWindow
WinDrawWindowFrame
WinDisableWindow
WinEnableWindow
WinGetWindowPointer
WinInitializeWindow
WinMoveWindowAddr
WinRemoveWindow
WinScreenInit
WinSetColors

B

Compatibility Guide

This appendix lists groups of functions and other features (such as events and launch codes) that have been added to the Palm OS® after version 1.0.

Before you use any new functions or features in an application, you must check to ensure that they are implemented in the OS version your application is running on. Checking the OS version number is *not* a reliable indicator that a specific feature is present, since some later OS versions do not include features present in earlier versions. In order to ensure that your code is supported, you must check for the presence of individual features.

To make this check easier, this appendix lists new functions and features in groups such that all functions and features in a group are always implemented together in the ROM of a Palm™ device. This means that you can check for a single feature in that group and be assured that if that feature is present than all functions and features in that group are implemented.

Each group includes a recommended test to check if it is implemented. The following groups are described:

2.0 New Feature Set 2382
3.0 New Feature Set 2385
3.1 New Feature Set 2389
3.2 New Feature Set 2392
International Feature Set 2394
Japanese Feature Set 2396
Wireless Internet Feature Set 2396
New Serial Manager Feature Set 2398
Connection Manager Feature Set. 2401
3.5 New Feature Set 2402
Notification Feature Set. 2408

4.0 New Feature Set 2409
Expansion Manager Feature Set 2415
VFS Manager Feature Set 2416
Bluetooth Library Feature Set 2418
High-Density Display Feature Set 2421
1.5x Display Feature Set 2423
Sound Stream Feature Set 2424
5.0 New Feature Set 2426
5.1 New Feature Set 2431
Graffiti 2 Feature Set 2436
5.3SC New Feature Set 2437
5.4 New Feature Set 2438
Bluetooth Sleep and Wake Scheduling Feature Set . . . 2439
ARM Debug Nub Feature Set 2440
Palm OS Cobalt Feature Set 2440
One-Handed Navigation Feature Set 2459

2.0 New Feature Set

You can check that this feature set is implemented by checking that the system version is 2.0 or higher. Use this `FtrGet()` call:

```
err = FtrGet(sysFtrCreator,
sysFtrNumROMVersion, &romversion);
```

The `romversion` parameter should be `0x02003000` or greater.

Launch Codes

This feature set adds the following launch codes:

`sysAppLaunchCmdLookup`
`sysAppLaunchCmdPanelCalledFromApp`
`sysAppLaunchCmdReturnFromPanel`
`sysAppLaunchCmdSystemLock`

Functions

This feature set adds the following functions:

CategoryInitialize()
CategorySetName()
DmDeleteCategory()
DmDatabaseProtect()
EvtAddUniqueEventToQueue()
EvtEventAvail()
EvtSysEventAvail()
FldGetNumberOfBlankLines()
FldGetScrollValues()
FldSetInsertionPoint()
FntGetScrollValues()
FntWordWrap()
FntWordWrapReverseNLines()
FrmPointInTitle()
FrmSetMenu()
FrmSetObjectBounds()
KeySetMask()
LocGetNumberSeparators()
LstScrollList()
LstGetVisibleItems()
MemCmp()
MenuSetActiveMenuRscID()
PhoneNumberLookup()
PrefSetPreference()
PrefGetPreference()
SclDrawScrollBar()
SclGetScrollBar()
SclHandleEvent()
SclSetScrollBar()
SerControl()
StrDelocalizeNumber()
StrLocalizeNumber()
StrNCaselessCompare()
StrNCat()
StrNCompare()
StrNCopy()
StrPrintF()
StrVPrintF()
SysBinarySearch()
SysCreateDataBaseList()
SysCreatePanelList()
SysErrString()
SysGraffitiReferenceDialog()
SysLibLoad()
SysStringByIndex()
SysTicksPerSecond()

TblHasScrollBar()
TblSetBounds()
TblSetColumnEditIndicator()
TblSetRowStaticHeight()

WinSetWindowBounds

Existing Functions that Changed

Several functions that existed in 1.0 were changed in 2.0:

CategoryCreateList() (old function renamed CategoryCreateListV10())

CategoryEdit() (old function renamed CategoryEditV10())

CategoryFreeList() (old function renamed CategoryFreeListV10())

CategorySelect() (old function renamed CategorySelectV10())

SelectDay() (old function renamed SelectDayV10())

DmFindSortPosition() (old function renamed DmFindSortPositionV10())

PrefGetAppPreferences() (old function renamed PrefGetAppPreferencesV10())

PrefOpenPreferenceDB (old function renamed PrefOpenPreferenceDBV10())

PrefSetAppPreferences() (old function renamed PrefSetAppPreferencesV10())

SerReceive() (old function renamed SerReceive10())

SerSend() (old function renamed SerSend10())

SysKeyboardDialog() (old function renamed SysKeyboardDialogV10())

Other Changes

As a rule, all Palm OS applications developed with the 1.0 SDK should run error-free on the latest device. There are two possible pitfalls for 1.0 applications:

- **fldChangedEvent Change—**The operating system now correctly sends a `fldChangedEvent` whenever a field object is changed. Previously, the event was at times not sent, especially when a `FldSetText` operation was performed. If your application doesn't catch the events that are now sent, it may have problems.
- **Non-standard tools—**If your application was not developed with Metrowerks Code Warrior for the Palm OS, it may run into problems. One known problem can occur if the application:
  - was compiled with optimization turned on
  - uses system preferences

3.0 New Feature Set

You can check that this feature set is implemented by checking that the system version is 3.0 or higher. Use this `FtrGet()` call:

```
err = FtrGet(sysFtrCreator,
sysFtrNumROMVersion, &romVersion);
```

The `romVersion` parameter should be greater than or equal to `0x03003000`, which can be constructed using the `sysMakeROMVersion` macro:

```
sysMakeROMVersion(3,0,0,sysROMStageRelease,0)
```

Launch Codes

This feature set adds the following launch codes:

`sysAppLaunchCmdExgAskUser`
`sysAppLaunchCmdExgReceiveData`

In addition, the launch code `sysAppLaunchCmdGoTo` is now also sent by the exchange manager, in addition to its use by the global find operation.

Font

This feature set adds the following font:

`largeBoldFont`

Functions

This feature set adds the following functions:

Dynamic User Interface Functions

`CtlNewControl()`
`CtlValidatePointer()`
`FldNewField()`
`FrmNewBitmap()`
`FrmNewForm()`
`FrmNewGadget()`
`FrmNewLabel()`
`FrmRemoveObject()`
`FrmValidatePtr()`
`LstNewList()`
`WinValidateHandle()`

For more information on creating and using dynamic user interface elements, see the section "Dynamic UI" on page 149 of the *Palm OS Programmer's Companion*, vol. I.

Font Functions

`FontSelect()`
`FntDefineFont()`

For more information on these functions and the support for custom fonts, see "Fonts" on page 276 of the *Palm OS Programmer's Companion*, vol. I.

Progress Manager Functions

`PrgHandleEvent()`
`PrgStartDialog()`
`PrgStopDialog()`
`PrgUpdateDialog()`
`PrgUserCancel()`

For more information, see the section "Progress Dialogs" on page 94 of the *Palm OS Programmer's Companion*, vol. I.

File Streaming Functions

FileClearerr()
FileClose()
FileControl()
FileDelete()
FileDmRead()
FileEOF()
FileError()
FileFlush()
FileGetLastError()
FileOpen()
FileRead()
FileReadLow() (system use only)
FileRewind()
FileSeek()
FileTell()
FileTruncate()
FileWrite()

For more information, see the section "File Streaming Application Program Interface" on page 210 of the *Palm OS Programmer's Companion*, vol. I.

Sound Functions

SndCreateMidiList()
SndPlaySmf()
SndDoCmd() (enhanced in 3.0)

Exchange Manager Functions

ExgAccept()
ExgDBRead()
ExgDBWrite()
ExgDisconnect()
ExgPut()
ExgReceive()
ExgRegisterData()
ExgSend()

For more information, see the chapter Beaming (Infrared Communication) in the *Palm OS Programmer's Companion*, vol. II, *Communications*.

IR Library Functions

IrAdvanceCredit()
IrBind()
IrClose()
IrConnectIrLap()
IrConnectReq()
IrConnectRsp()
IrDataReq()
IrDisconnectIrLap()
IrDiscoverReq()
IrIAS_Add()
IrIAS_GetInteger()
IrIAS_GetIntLsap()
IrIAS_GetObjectID()
IrIAS_GetOctetString()
IrIAS_GetOctetStringLen()
IrIAS_GetType()
IrIAS_GetUserString()
IrIAS_GetUserStringCharSet()
IrIAS_GetUserStringLen()
IrIAS_Next()
IrIAS_Query()
IrIAS_SetDeviceName()
IrIAS_StartResult()
IrIsIrLapConnected()
IrIsMediaBusy()
IrIsNoProgress()
IrIsRemoteBusy()
IrLocalBusy()
IrMaxRxSize()
IrMaxTxSize()
IrOpen()
IrSetConTypeLMP()
IrSetConTypeTTP()
IrSetDeviceInfo()
IrTestReq()
IrUnbind()

For more information, see the chapter Beaming (Infrared Communication) in the *Palm OS Programmer's Companion*, vol. II, *Communications*.

Miscellaneous Functions

FrmRestoreActiveState()
FrmSaveActiveState()
ScrDisplayMode()
SysGetAppInfo() (system use only)
SysGetOSVersionString()
SysGetROMToken()
SysGetStackInfo()
SysGremlins()
TblGetItemFont()
TblSetItemFont()

Existing Functions that Changed

Two functions that existed in 2.0 were changed in 3.0:

CategoryEdit() (old function renamed CategoryEditV20())

SysBatteryInfo() (old function renamed SysBatteryInfoV20())

Other Changes

- The dynamic heap has been increased in size to 96 KB.
- Storage RAM is no longer subdivided into multiple storage heaps of 64 KB each. All storage RAM on a memory card is configured as a single storage heap.
- Each flash ROM-based Palm device holds a serial number that identifies it uniquely and can be retrieved via `SysGetROMToken()`. For more information, see "Retrieving the ROM Serial Number" on page 366 of the *Palm OS Programmer's Companion*, vol. I.
- The Application Launcher (accessed via the silkscreen "Applications" button) is now an application, rather than a popup. The `SysAppLauncherDialog` function, which provides the API to the old popup launcher, is still present in Palm OS 3.0 for compatibility purposes, but has not been updated and generally should not be used. For more information, see "Application Launcher" on page 160 of the *Palm OS Programmer's Companion*, vol. I.
- The sound manager supports MIDI sound files, adding new sounds, asynchronous playback, and other features. There are also new selectors for setting the volume preferences. For more information, see the section "System Boot and Reset" on page 361 of the *Palm OS Programmer's Companion*, vol. I.

The following functions existed in the system previously, but were not documented:

`RctCopyRectangle()`	`RctOffsetRectangle()`
`RctGetIntersection()`	`RctPtInRectangle()`
`RctInsetRectangle()`	`RctSetRectangle()`

The following event type existed in the system previously, but was not previously documented:

`frmGotoEvent`

3.1 New Feature Set

You can check that this feature set is implemented by checking that the system version is 3.1 or higher. Use this `FtrGet()` call:

```
err = FtrGet(sysFtrCreator,
sysFtrNumROMVersion, &romVersion);
```

The `romVersion` parameter should be greater than or equal to `0x03103000`, which can be constructed using the `sysMakeROMVersion` macro:

```
sysMakeROMVersion(3,1,0,sysROMStageRelease,0)
```

Functions

This feature set adds the following functions:

`ChrHorizEllipsis()`
`ChrNumericSpace()`
ContrastAdjust
`FntWidthToOffset()`
`FtrPtrNew()`
`FtrPtrFree()`
`FtrPtrResize()`
`SelectOneTime()`
`WinDrawChar()`
`WinDrawTruncChars()`

NOTE: The `PalmOSGlue.lib` provides compatibility functions and macros for `ChrHorizEllipsis`, `ChrNumericSpace`, `WinDrawChar`, and `WinDrawTruncChars.` If you want to use these functions on systems that don't have the 3.1 feature set, you can link your application with `PalmOSGlue.lib`. See the chapter "PalmOSGlue Library" on page 1929 for more information.

Changes to the Character Encoding

Starting in Palm OS 3.1, the character encoding used on most systems is Microsoft Windows code page 1252. Versions prior to 3.1 used an encoding that was very similar to code page 1252 but did not follow it exactly. The following changes to the character set are introduced in Palm OS 3.1:

- Some of the special Palm OS glyphs in the high ASCII range (such as the shortcut stroke and the command stroke) have been moved down into the control code range, and other characters (such as the numeric space and horizontal ellipsis) have been copied into the control range so that they're guaranteed to exist in every encoding. For the numeric space and horizontal ellipsis, you can use the macros `ChrNumericSpace()` and `ChrHorizEllipsis()` to return the appropriate character regardless of the character map. In `PalmOSGlue.lib`, these two macros are named `TxtGlueGetNumericSpaceChar` and `TxtGlueGetHorizEllipsisChar`, respectively.
- The four playing-card characters have been moved from the high ASCII range in the standard four fonts to the 9-point Symbol font.

Other Changes in 3.1

- Palm OS 3.1 supports a new processor: the EZ Dragonball processor. This processor is compatible with the existing Dragonball processor, so your application should run without changes as long as it doesn't access registers or system globals directly.

  If your application needs to know if it is running on an EZ Dragonball, it can check using the following code:

```
DWord id, chip;
Word revision;
Err err;
err = FtrGet(sysFtrCreator,
      sysFtrNumProcessorID, &id);
if (!err) {
  chip = id & sysFtrNumProcessorMask;
  revision = id & 0x0ffff;
  if (chip==sysFtrNumProcessor328)
    // traditional Dragonball
  else if (chip==sysFtrNumProcessorEZ)
    // Dragonball EZ
}
```

- The constant `preferenceDataVersion` was removed and replaced with `preferenceDataVerLatest`.

- Character variables are now two bytes long. The type `WChar` defines a character variable.
- The `keyDownEvent` structure's `chr` field (which contains the input character) has been changed from a `Word` to a `WChar`.
- The string manager functions `StrChr()` and `StrStr()` now treat buffers as characters, not arbitrary byte arrays. If you previously used these functions to search data buffers, your code may no longer work.
- The string manager function `StrToLower()` can now handle any type of characters, including accented characters.
- The underline attribute of `FieldAttrType` now has support for the value 2. Previously, the only underline modes available were no underline (0) and gray underline (1). In Palm OS 3.1 and higher, the value 2 is interpreted as solid underline. The `UnderlineModeType` enum defined in `Window.h` defines the possible values for the underline attribute.
- The use of the `DmGetNextDatabaseByTypeCreator()` `onlyLatestVers` parameter changed in 3.1. If `onlyLatestVers` is `true`, you only receive one matching database for each type/creator pair. In version 3.0 and earlier, you could receive multiple matching databases if `onlyLatestVers` was `true`. See that function's description for a more detailed description.

3.2 New Feature Set

You can check that this feature set is implemented by checking that the system version is 3.2 or higher. Use this `FtrGet()` call:

```
err = FtrGet(sysFtrCreator, sysFtrNumROMVersion,
&romVersion);
```

The `romVersion` parameter should be greater than or equal to `0x03203000`, which can be constructed using the `sysMakeROMVersion` macro:

```
sysMakeROMVersion(3,2,0,sysROMStageRelease,0)
```

Functions

This feature set adds the following functions:

AlmGetProcAlarm()	NetLibConfigIndexFromName()
AlmSetProcAlarm()	NetLibConfigList()
ClipboardAppendItem()	NetLibConfigMakeActive()
DmGetDatabaseLockState()	NetLibConfigRename()
ErrAlert()	NetLibConfigSaveAs()
NetLibConfigAliasGet()	NetLibOpenConfig()
NetLibConfigAliasSet()	SndPlaySmfResource()
NetLibConfigDelete()	

Note that the `NetLib...` functions, although present in the 3.2 New Feature Set, are first declared in the Palm OS Garnet SDK.

Existing Functions that Changed

Two functions that existed in 3.0 were changed in 3.2:

SysGremlins() was removed and replaced with a `SysGremlins` macro that maps it to the function `HostGremlinIsRunning`. The prototype is slightly different, but you can still call `SysGremlins` in the same way you did before.

PrgStartDialog() (old function renamed PrgStartDialogV31())

Other Changes in 3.2

- The prototype for the system use only function `AlmDisplayAlarm` changed from no return value to a Boolean return value. This change may affect system patches and extensions that intercept `AlmDisplayAlarm` calls.

International Feature Set

You can check that this feature set is implemented by checking for the existence of the international manager. You can check by calling `FtrGet()` as follows:

```
err = FtrGet(sysFtrCreator, sysFtrNumIntlMgr, &value);
```

If the international manager is installed, the `value` parameter will be non-zero and the returned error should also be zero (for no error).

You can learn more about the international manager by reading the chapter "Localized Applications" on page 385 in the *Palm OS Programmer's Companion*, vol. I.

NOTE: If you want to use international functions on systems that don't have the international feature, you can link your application with `PalmOSGlue.lib`. The functions in this library are the same as those listed below except that they use the prefix "TxtGlue" instead of "Txt." For more information, see the chapter "PalmOSGlue Library" on page 1929.

Functions

This feature set adds the following functions:

Text Manager Functions

TxtByteAttr()
TxtCaselessCompare()
TxtCharAttr()
TxtCharBounds()
TxtCharEncoding()
TxtCharIsAlNum()
TxtCharIsAlpha()
TxtCharIsCntrl()
TxtCharIsDigit()
TxtCharIsGraph()
TxtCharIsHardKey()
TxtCharIsHex()
TxtCharIsLower()
TxtCharIsPrint()
TxtCharIsPunct()
TxtCharIsSpace()
TxtCharIsUpper()
TxtCharSize()
TxtCharXAttr()
TxtCompare()
TxtEncodingName()
TxtFindString()
TxtGetChar()
TxtGetNextChar()
TxtGetPreviousChar()
TxtCharIsValid()
TxtMaxEncoding()
TxtNextCharSize()
TxtPreviousCharSize()
TxtReplaceStr()
TxtSetNextChar()
TxtStrEncoding()
TxtTransliterate()
TxtGetTruncationOffset()
TxtWordBounds()

Other Functions

IntlGetRoutineAddress()

Removed Functions and Macros

If the international feature set exists, then the following functions and macros are no longer available:

```
GetCharAttr()
GetCharCaselessValue()
GetCharSortValue()
IsAscii()
IsAlNum()
IsAlpha()
IsCntrl()
IsDigit()
IsGraph()
IsLower()
IsPrint()
IsPunct()
IsSpace()
IsUpper()
IsHex()
IsDelim()
```

Japanese Feature Set

You can check that the Japanese feature set is implemented by checking if the unit is Japanese. You can check by calling `FtrGet()` as follows:

```
err = FtrGet(sysFtrCreator, sysFtrNumEncoding, &value);
```

The unit has the Japanese OS if the `value` parameter is `charEncodingCP932`.

For further information about the Japanese implementation, see the section "Notes on the Japanese Implementation" in the *Palm OS Programmer's Companion*, vol. I.

Wireless Internet Feature Set

You can check that this feature set is implemented by checking for the existence of the Web Clipping Application Viewer (**Viewer**) and iMessenger™ applications. Here's an example of how to check for Viewer:

```
DmSearchStateType searchState;
UInt16 cardNo;
LocalID dbID;
err = DmGetNextDatabaseByTypeCreator(true,
&searchState, sysFileTApplication,
sysFileCClipper, true, &cardNo, &dbID);
```

If Viewer is not present, the `DmGetNextDatabaseByTypeCreator` routine returns an error. To check for iMessenger, you can use the creator type `sysFileCMessaging`.

NOTE: The Viewer was formerly described as the Clipper.

You can learn more about the Palm.Net™ system for wireless Internet access and the programmatic interfaces to the Viewer and iMessenger applications by reading the chapter "Internet and Messaging Applications" in the *Palm OS Programmer's Companion*, vol. II, *Communications*. For a more complete description, see the *Web Clipping Developer's Guide*.

Launch Codes

This feature set adds the following launch codes:

`sysAppLaunchCmdAddRecord` (for iMessenger application; existed for Mail in 3.0)
`sysAppLaunchCmdGoToURL`
`sysAppLaunchCmdOpenDB`
`sysAppLaunchCmdURLParams`

Events

This feature set adds the following events:

`inetSockReadyEvent`
`inetSockStatusChangeEvent`

This feature set also adds the following `keyDownEvent` key codes:

```
vchrHardAntenna
vchrRadioCoverageOK
vchrRadioCoverageFail
```

These key codes are described in the section Wireless keyDownEvent Key Codes.

Functions

This feature set adds the following functions.

Internet Library Functions

For more information, see the chapter "Network Communication" in the *Palm OS Programmer's Companion*, vol. II, *Communications*.

INetLibCacheGetObject()
INetLibCacheList()
INetLibCheckAntennaState()
INetLibClose()
INetLibConfigAliasGet()
INetLibConfigAliasSet()
INetLibConfigDelete()
INetLibConfigIndexFromName()
INetLibConfigList()
INetLibConfigMakeActive()
INetLibConfigRename()
INetLibConfigSaveAs()
INetLibGetEvent()
INetLibOpen()
INetLibSettingGet()
INetLibSettingSet()
INetLibSockClose()
INetLibSockConnect()
INetLibSockHTTPAttrGet()
INetLibSockHTTPAttrSet()
INetLibSockHTTPReqCreate()
INetLibSockHTTPReqSend()
INetLibSockOpen()
INetLibSockRead()
INetLibSockSettingGet()
INetLibSockSettingSet()
INetLibSockStatus()
IINetLibURLCrack
INetLibURLGetInfo()
INetLibURLOpen()
INetLibURLsAdd()
INetLibWiCmd()

New Serial Manager Feature Set

The New Serial Manager feature set has two different versions.

New Serial Manager Feature Set Version 1

You can check that this feature set is implemented by checking for the existence of the new Serial Manager. You can check by calling FtrGet() as follows:

```
err = FtrGet(sysFileCSerialMgr,
sysFtrNewSerialPresent, &value);
```

If the new Serial Manager is installed, the `value` parameter will be non-zero and the returned error should also be zero (for no error).

You can learn more about the new Serial Manager and Connection Manager by reading the sections "The Serial Manager" on page 92 and "The Connection Manager" on page 116 in the *Palm OS Programmer's Companion*, vol. II, *Communications*.

This feature set adds the following functions.

Serial Manager Functions

SrmClearErr()
SrmClose()
SrmControl()
SrmGetDeviceCount()
SrmGetDeviceInfo()
SrmGetStatus()
SrmOpen()
SrmOpenBackground()
SrmPrimeWakeupHandler()
SrmReceive()
SrmReceiveCheck()
SrmReceiveFlush()
SrmReceiveWait()
SrmReceiveWindowClose()
SrmReceiveWindowOpen()
SrmSend()
SrmSendCheck()
SrmSendFlush()
SrmSendWait()
SrmSetReceiveBuffer()
SrmSetWakeupHandler()
WakeupHandlerProcPtr()

Virtual Driver Functions

DrvEntryPointProcPtr()
GetSizeProcPtr()
GetSpaceProcPtr()
VdrvControlProcPtr()
VdrvOpenProcPtr()
VdrvStatusProcPtr()
VdrvWriteProcPtr()
WriteBlockProcPtr()
WriteByteProcPtr()

Connection Manager Functions

CncAddProfile()
CncDeleteProfile()
CncGetProfileInfo()
CncGetProfileList()

Serial Link Manager Function

SlkSocketPortID()

New Serial Manager Feature Set Version 2

You can check that version 2 of the new Serial Manager feature set is implemented by checking the Serial Manager version number and the Palm OS version number. You can check by calling FtrGet() as follows:

```
err = FtrGet(sysFileCSerialMgr,
  sysFtrNewSerialVersion, &value);
err = FtrGet(sysFtrCreator,
  sysFtrNumROMVersion, &romVersion);
```

The new Serial Manager is present if:

- Both calls to `FtrGet` return zero (for no error).
- The `value` parameter is 2.
- The `romVersion` parameter is `0x04003000`, which can be constructed using the `sysMakeROMVersion` macro:

```
sysMakeROMVersion(4,0,0,sysROMStageRelease,0)
```

This feature set adds the following functions.

Serial Manager Functions

SrmCustomControl()
SrmExtOpen()
SrmExtOpenBackground()

Virtual Driver Functions

VdrvControlCustomProcPtr()
VdrvOpenProcV4Ptr()
SignalCheckPtr()

IMPORTANT: Some Handspring devices ship with Palm OS version 3.5 but have new Serial Manager feature set version 2. These devices support the `SignalCheckPtr` virtual driver function and have expanded functionality for USB support, but they do not support the other function calls in this feature set.

You can learn more about the new Serial Manager by reading the chapter "Serial Communication" on page 89 of *Palm OS Programmer's Companion*, vol. II, *Communications*.

Connection Manager Feature Set

You can check that the Connection Manager feature set is implemented by checking the value of the Connection Manager feature. You can check by calling `FtrGet()` as follows:

```
err = FtrGet(kCncFtrCncMgrCreator,
kCncFtrCncMgrVersion, &version);
```

The `version` parameter should be greater than or equal to `0x00040001`. In the 4.0 Palm OS SDK, this value is represented by the `kCncMgrVersion` constant.

NOTE: An earlier version of the Connection Manager is available if New Serial Manager Feature Set Version 1 is present.

Functions

This feature set adds the following functions:

`CncProfileCloseDB()`
`CncProfileCount()`
`CncProfileCreate()`
`CncProfileDelete()`
`CncProfileGetCurrent()`
`CncProfileGetIDFromIndex()`
`CncProfileGetIDFromName()`
`CncProfileGetIndex()`
`CncProfileOpenDB()`
`CncProfileSetCurrent()`
`CncProfileSettingGet()`
`CncProfileSettingSet()`

3.5 New Feature Set

You can check that this feature set is implemented by checking that the system version is 3.5 or higher. Use this FtrGet() call:

```
err = FtrGet(sysFtrCreator,
sysFtrNumROMVersion, &romVersion);
```

The romVersion parameter should be greater than or equal to 0x03503000, which can be constructed using the sysMakeROMVersion macro:

```
sysMakeROMVersion(3,5,0,sysROMStageRelease,0)
```

Launch Codes

This feature set adds the following launch codes:

sysAppLaunchCmdNotify

Events

This feature set adds the following events:

frmGadgetEnterEvent
frmGadgetMiscEvent
menuCmdBarOpenEvent
menuOpenEvent

Functions

This feature set adds the following functions:

Bitmaps

BmpBitsSize()
BmpColortableSize()
BmpCompress()
BmpCreate()
BmpDelete()
BmpGetBits()
BmpGetColortable()
BmpSize()
ColorTableEntries()

For more information on creating and using bitmaps, see the section "Bitmaps" on page 131 of the *Palm OS Programmer's Companion*, vol. I.

Controls

CtlGetSliderValues()
CtlNewGraphicControl()
CtlNewSliderControl()
CtlSetGraphics()
CtlSetSliderValues()

These functions add support for graphical buttons and slider controls. For more information, see the section "Offscreen Windows" on page 99 of the *Palm OS Programmer's Companion*, vol. I.

Forms

FrmCustomResponseAlert()
FrmNewGsi()
FrmSetGadgetHandler()

Among the changes to form functions is extended gadget support. For more information on gadgets and extended gadgets, see the section "Custom UI Objects (Gadgets)" on page 146 of the *Palm OS Programmer's Companion*, vol. I.

Menus

MenuAddItem()
MenuCmdBarDisplay()
MenuHideItem()
MenuCmdBarAddButton()
MenuCmdBarGetButtonData()
MenuShowItem()

For more information on using menu functions, see the section "Menus" on page 113 of the *Palm OS Programmer's Companion*, vol. I.

Overlay Manager

OmGetCurrentLocale()
OmGetIndexedLocale()
OmGetRoutineAddress()
OmGetSystemLocale()
OmLocaleToOverlayDBName()
OmOverlayDBNameToLocale()
OmSetSystemLocale()

For more information on using the overlay manager, see the section "Using Overlays to Localize Resources" on page 387 of the *Palm OS Programmer's Companion*, vol. I.

Private Records

`SecSelectViewStatus()`
`SecVerifyPW()`

Tables

`TblGetItemPtr()`
`TblSetColumnMasked()`
`TblRowMasked()`
`TblSetRowMasked()`

UI Colors

`UIColorGetTableEntryIndex()`
`UIColorSetTableEntry()`
`UIColorGetTableEntryRGB()`

For more information on using the UI Colors API, see the section "Color and Grayscale Support" on page 151 of the *Palm OS Programmer's Companion*, vol. I.

UI Controls

`UIBrightnessAdjust()`
`UIPickColor()`

Windows

WinCreateBitmapWindow()
WinDrawPixel()
WinErasePixel()
WinGetBitmap()
WinGetPatternType()
WinGetPixel()
WinIndexToRGB()
WinInvertPixel()
WinPaintBitmap()
WinPaintChar()
WinPaintChars()
WinPaintLine()
WinPaintLines()
WinPaintPixel()
WinPaintPixels()
WinPaintRectangle()
WinPaintRectangleFrame()
WinPalette()
WinPopDrawState()
WinPushDrawState()
WinRGBToIndex()
WinScreenLock()
WinScreenMode()
WinScreenUnlock()
WinSetBackColor()
WinSetDrawMode()
WinSetForeColor()
WinSetPatternType()
WinSetTextColor()

For more information on using the window functions, see the section "Drawing on the Palm Powered Handheld" on page 77 of the *Palm OS Programmer's Companion*, vol. I.

Miscellaneous New Functions

DmOpenDBNoOverlay()
ExgDoDialog()
DateToAscii()
ResLoadConstant()
TxtParamString()

Existing Functions that Changed

The following functions that existed prior to 3.5 have changed in release 3.5:

`ScrDisplayMode` was changed to WinScreenMode().

`ContrastAdjust` was changed to UIContrastAdjust().

SelectTime() (old function renamed SelectTimeV33())

New Data Types

The data types `Byte`, `Word`, `DWord` and so on are now deprecated. It is recommend that you use the corresponding new data types. For

example, use `Int16` instead of `SWord` and `UInt32` instead of `DWord`. In particular, the unfortunate distinction between `Handle`/`VoidHand` has been fixed; use `MemHandle` instead.

To learn in general how the type names changed, see the header file `PalmOSCompatibility.h`. This file provides a mapping from the old type name to the new type name. If you need to move forward without modifying your code, you can include this file in your project to provide declarations for the old type names.

Changes to Events

- The `tapCount` field has been added to the `EventType` structure. The `tapCount` field specifies the number of times the user tapped the pen at the current location; in fields, two taps selects a word, and three taps selects a line.

IMPORTANT: Because the `tapCount` field has been added to the `EventType` structure, it has become more critical that you clear the event structure before using it to add a new event to the queue. Otherwise, the `tapCount` will be incorrect for the new event.

- The structures for `ctlRepeatEvent` and `ctlSelectEvent` have a `value` field added to them. This new field is used only for sliders; it holds the current value of the slider.
- Form objects now handle the `frmTitleSelectEvent` by adding a `keyDownEvent` with the `vchrMenu` character to the event queue (which causes the form's menu to display).
- Some of the structure definitions for system-level events have moved from `Event.h` to `SysEvent.h`.
- The `winEnterEvent` is now not generated until `FrmDrawForm()` is called. Make sure to draw your form in response to `frmOpenEvent`, not `winEnterEvent`.
- `EvtSetNullEventTick()` is now a function. In previous releases it was a macro.

Other Changes

- `FrmDrawForm()`

  On release 3.5, `FrmDrawForm` erases the window's rectangle before it draws, so you must perform custom drawing after the call to `FrmDrawForm`, not before. If you have drawn before the call to `FrmDrawForm`, your changes are lost. On debug ROMs, the window handle is invalid until `FrmDrawForm` is called so that draws before `FrmDrawForm` result in a bus error.

- Resource Manager

  The resource manager functions have been updated to work with overlay databases. See "Using Overlays to Localize Resources" on page 387 of the *Palm OS Programmer's Companion*, vol. I.

- `DmGetDatabase()`

  The order in which this call returns databases has changed. Previously all of the databases from ROM were returned first, then all from RAM. Now they are intermingled. Developers should not rely on the order in which databases are returned from this call.

- `StrToLower()`

  This function is different in 3.5 Latin ROMs. Previously it only changed A through Z. Now it also changes high ASCII characters.

- Time Manager

  If you are using a debug ROM, the string buffer is filled with `dateStringLength` or `longStrLength` debugging bytes, depending on the `dateFormat` parameter. For the routines that return the day-of-week name in addition to the date, the size of the buffers has been expanded, so developers need to check the max lengths defined in `DateTime.h`.

- The format of the storage heap header has changed, thus any existing saved Simulator card images are invalid and should be tossed.

- Category Data Structures

  The data structure `AppInfoType` has been documented.

  `CategoryCreateList()` now has a "hide" function with two new constants; `categoryHideEditCategory, and categoryDefaultEditCategoryString`.

- `FtrPtrNew()`

  `FtrPtrNew` now allows allocating chunks larger than 64KB.

- Dynamic heap

  The dynamic heap is now sized based on the amount of memory available to the system.

Table B.1 Dynamic heap size

Device RAM Size	Heap Size
$x < 2MB$	64KB
$2MB \leq x < 4MB$	128KB
$x \geq 4MB$	256KB

Notification Feature Set

You can check that this feature set is implemented by checking for the existence of the notification manager. You can check by calling `FtrGet()` as follows:

```
err = FtrGet(sysFtrCreator,
sysFtrNumNotifyMgrVersion, &value);
```

If the notification manager is part of the system, the `value` parameter will be non-zero and the returned error should also be zero (for no error).

Notification Manager

`SysNotifyBroadcast()`
`SysNotifyBroadcastDeferred()`
`SysNotifyRegister()`
`SysNotifyUnregister()`

To learn more about the notification manager, see the section "Notifications" on page 30 of the *Palm OS Programmer's Companion*, vol. I.

4.0 New Feature Set

You can check that this feature set is implemented by checking that the system version is 4.0 or higher. Use this `FtrGet()` call:

```
err = FtrGet(sysFtrCreator,
sysFtrNumROMVersion, &romVersion);
```

The `romVersion` parameter should be greater than or equal to `0x04003000`, which can be constructed using the `sysMakeROMVersion` macro:

```
sysMakeROMVersion(4,0,0,sysROMStageRelease,0)
```

Launch Codes

This feature set adds the following launch codes:

`sysAppLaunchCmdAttention`

`sysAppLaunchCmdExgGetData`

`sysAppLaunchCmdExgPreview`

Notifications

This feature set adds the following notifications:

cncNotifyProfileEvent

sysExternalConnectorAttachEvent

sysExternalConnectorDetachEvent

sysNotifyCardInsertedEvent

sysNotifyCardRemovedEvent

sysNotifyDBDeletedEvent

sysNotifyDeleteProtectedEvent

sysNotifyDeviceUnlocked

sysNotifyGotUsersAttention

sysNotifyHelperEvent

sysNotifyLocaleChangedEvent

sysNotifyNetLibIFMediaEvent

sysNotifyRetryEnqueueKey

sysNotifyVolumeMountedEvent

sysNotifyVolumeUnmountedEvent

Functions

This feature set adds the following functions:

Attention Manager

AttnDoSpecialEffects()
AttnForgetIt()
AttnGetAttention()
AttnGetCounts()
AttnIndicatorEnable()
AttnIndicatorEnabled()
AttnIterate()
AttnListOpen()
AttnUpdate()

Date and Time Manager

TimeZoneToAscii()

TimTimeZoneToUTC()

TimUTCToTimeZone()

Exchange Manager

ExgGetTargetApplication()
ExgGetDefaultApplication()
ExgGetRegisteredApplications()
ExgSetDefaultApplication()
ExgControl()
ExgNotifyGoto()
ExgNotifyPreview()
ExgRequest()
ExgGetRegisteredTypes()

For more information on the Exchange Manager, see the chapter Chapter 1, "Object Exchange," on page 1 of the *Palm OS Programmer's Companion*, vol. II, *Communications*.

Exchange Library

ExgLibAccept()
ExgLibClose()
ExgLibConnect()
ExgLibControl()
ExgLibDisconnect()
ExgLibGet()
ExgLibHandleEvent()
ExgLibOpen()
ExgLibPut()
ExgLibReceive()
ExgLibRequest()
ExgLibSend()
ExgLibSleep()
ExgLibWake()

For more information on the Exchange Library API, see Chapter 63, "Exchange Library," on page 1409 of the *Palm OS Programmer's Companion*, vol. II, *Communications*.

Locale Manager

LmGetLocaleSetting()
LmGetNumLocales()
LmLocaleToIndex()

Miscellaneous UI

PhoneNumberLookupCustom()

Notification Manager

SysNotifyBroadcastFromInterrupt()

PDI Library Functions

PdiDefineReaderDictionary()
PdiDefineResizing()
PdiEnterObject()
PdiLibClose()
PdiLibOpen()
PdiParameterPairTest()
PdiReaderDelete()
PdiReaderNew()
PdiReadParameter()
PdiReadProperty()
PdiReadPropertyField()
PdiReadPropertyName()
PdiSetEncoding()
PdiWriteBeginObject()
PdiWriteEndObject()
PdiWriteParameter()
PdiWriteParameterStr()
PdiWriteProperty()
PdiWritePropertyBinaryValue()
PdiWritePropertyFields()
PdiWritePropertyStr()
PdiWritePropertyValue()
PdiWriterDelete()
PdiWriterNew()

For more information, see the chapter Chapter 3, "Personal Data Interchange," in *Palm OS Programmer's Companion*, vol. II, *Communications*.

Sound Manager Functions

SndInterruptSmfIrregardless()
SndPlaySmfResourceIrregardless()
SndPlaySmfIrregardless()

For more information, see the chapter Chapter 11, "Palm System Support," in *Palm OS Programmer's Companion*, vol. I.

Telephony Manager Functions

TelCancel()
TelCfgGetPhoneNumber()
TelCfgGetSmsCenter()
TelCfgSetSmsCenter()
TelClose()
TelClosePhoneConnection()
TelDtcCallNumber()
TelDtcCloseLine()
TelDtcReceiveData()
TelDtcSendData()
TelEmcCall()
TelEmcCloseLine()
TelEmcGetNumber()
TelEmcGetNumberCount()
TelEmcSelectNumber()
TelEmcSetNumber()
TelGetCallState()
TelGetEvent()
TelGetTelephonyEvent()
TelInfGetInformation()
TelIs<FunctionName>Supported()
TelIs<ServiceSet>Available()
TelIsCfgServiceAvailable()
TelIsDtcServiceAvailable()
TelIsEmcServiceAvailable()
TelIsInfServiceAvailable()
TelIsNwkServiceAvailable()
TelIsOemServiceAvailable()
TelIsPhbServiceAvailable()
TelIsPhoneConnected()
TelIsPowServiceAvailable()
TelIsSmsServiceAvailable()
TelIsSndServiceAvailable()
TelIsSpcServiceAvailable()
TelIsStyServiceAvailable()
TelMatchPhoneDriver()

TelNwkGetLocation()
TelNwkGetNetworkName()
TelNwkGetNetworks()
TelNwkGetNetworkType()
TelNwkGetSearchMode()
TelNwkGetSelectedNetwork()
TelNwkGetSignalLevel()
TelNwkSelectNetwork()
TelNwkSetSearchMode()
TelOemCall()
TelOpen()
TelOpenPhoneConnection()
TelPhbAddEntry()
TelPhbDeleteEntry()
TelPhbGetAvailablePhonebooks()
TelPhbGetEntries()
TelPhbGetEntry()
TelPhbGetEntryCount()
TelPhbGetEntryMaxSizes()
TelPhbGetSelectedPhonebook()
TelPhbSelectPhonebook()
TelPowGetBatteryStatus()
TelPowGetPowerLevel()
TelPowSetPhonePower()
TelSendCommandString()

TelSmsDeleteMessage()
TelSmsGetAvailableStorage()
TelSmsGetDataMaxSize()
TelSmsGetMessageCount()
TelSmsGetSelectedStorage()
TelSmsGetUniquePartId()
TelSmsReadMessage()
TelSmsReadMessages()
TelSmsReadReport()
TelSmsReadReports()
TelSmsReadSubmittedMessage()
TelSmsReadSubmittedMessages()
TelSmsSelectStorage()
TelSmsSendManualAcknowledge()
TelSmsSendMessage()
TelSndMute()
TelSndPlayKeyTone()
TelSndStopKeyTone()
TelSpcAcceptCall()
TelSpcCallNumber()
TelSpcCloseLine()
TelSpcConference()
TelSpcGetCallerNumber()
TelSpcHoldLine()
TelSpcPlayDTMF()
TelSpcRejectCall()
TelSpcRetrieveHeldLine()
TelSpcSelectLine()
TelSpcSendBurstDTMF()
TelSpcStartContinuousDTMF()
TelSpcStopContinuousDTMF()
TelStyChangeAuthenticationCode()
TelStyEnterAuthenticationCode()
TelStyGetAuthenticationState()

For more information about the Telephony Manager, see Chapter 10, "Telephony Manager," in *Palm OS Programmer's Companion*, vol. II, *Communications*.

Windows Manager

WinGetPixelRGB()
WinSetForeColorRGB()
WinSetBackColorRGB()
WinSetTextColorRGB()

Internationalization Functions

The following functions have been added to aid with creating localized applications:

FntWCharWidth()
StrCompareAscii()
TxtGetWordWrapOffset()
IntlSetRoutineAddress()
OmGetNextSystemLocale()
TxtConvertEncoding()
TxtNameToEncoding()

Existing Functions that Changed

The following functions that existed prior to 4.0 have changed in release 4.0:

ExgRegisterData() has been deprecated. The preferred function is now ExgRegisterDatatype().

ExgGet() was previously unimplemented. It has been implemented in release 4.0

ExgNotifyReceive() was previously a private function. It is now a public function, intended to be used by exchange libraries.

FldRecalculateField() now updates the word-wrapping information whenever `FldRecalculateField` is called, regardless of the value of the `redraw` parameter. Prior to Palm OS 4.0 it updated the word-wrapping information only if the `redraw` parameter was set to `true`.

StrNCaselessCompare() now requires both of its string parameters to be null-terminated.

StrNCompare() now requires both of its string parameters to be null-terminated.

Expansion Manager Feature Set

Because not every system has (or needs) Expansion Manager services, applications wishing to use these services should check to make sure they are present before calling them. This is accomplished by checking for the Expansion Manager's system feature with a call to FtrGet(), supplying `sysFileCExpansionMgr` for the feature creator and `expFtrIDVersion` for the feature number.

The following code shows how to check for the presence and proper version of the Expansion Manger. Note that `expectedExpMgrVersionNum` should be replaced by the actual version number you expect.

```
UInt32 expMgrVersion;
Err err;

err = FtrGet(sysFileCExpansionMgr,
  expFtrIDVersion, &expMgrVersion);
if(err){
    // Expansion Manager not installed
} else {
    // check version number of Expansion Manager,
    // if necessary
    if(expMgrVersion == expectedExpMgrVersionNum)
        // everything is OK
}
```

You can learn more about the expansion manager by reading Chapter 33, "Expansion Manager," on page 701.

Functions

This feature set adds the following functions.

Expansion Manager Functions

`ExpCardGetSerialPort()`
`ExpCardInfo()`
`ExpCardPresent()`
`ExpSlotDriverInstall()`
`ExpSlotDriverRemove()`
`ExpSlotEnumerate()`
`ExpSlotLibFind()`

VFS Manager Feature Set

Because not every system has (or needs) Virtual File System (VFS) Manager services, applications wishing to use these services should check to make sure they are present before calling them. This is accomplished by checking for the VFS Manager's system feature with a call to `FtrGet()`, supplying `sysFileCVFSMgr` for the feature creator and `vfsFtrIDVersion` for the feature number.

The following code shows how to check for the presence and proper version of the VFS Manger. Note that `expectedVFSMgrVersionNum` should be replaced by the actual version number you expect.

```
UInt32 vfsMgrVersion;
Err err;

err = FtrGet(sysFileCVFSMgr,
  vfsFtrIDVersion, &vfsMgrVersion);
if(err){
    // VFS Manager not installed
} else {
    // check version number of VFS Manager,
    // if necessary
    if(vfsMgrVersion == expectedVFSMgrVersionNum)
        // everything is OK
}
```

You can learn more about the VFS manager by reading Chapter 58, "Virtual File System Manager," on page 1143.

Functions

This feature set adds the following functions.

VFS Manager Functions

VFSCustomControl()	VFSFileSize()
VFSCustomControl()	VFSFileTell()
VFSDirEntryEnumerate()	VFSFileWrite()
VFSExportDatabaseToFile()	VFSGetDefaultDirectory()
VFSExportDatabaseToFileCustom()	VFSImportDatabaseFromFile()
VFSFileClose()	VFSImportDatabaseFromFileCustom()
VFSFileCreate()	VFSInstallFSLib()

VFSFileDBGetRecord()
VFSFileDBGetResource()
VFSFileDBInfo()
VFSFileDelete()
VFSFileEOF()
VFSFileGetAttributes()
VFSFileGetDate()
VFSFileOpen()
VFSFileRead()
VFSFileReadData()
VFSFileRename()
VFSFileResize()
VFSFileSetAttributes()
VFSFileSetDate()
VFSFileSeek()
VFSRegisterDefaultDirectory()
VFSRemoveFSLib()
VFSUnregisterDefaultDirectory()
VFSVolumeEnumerate()
VFSVolumeFormat()
VFSVolumeGetLabel()
VFSVolumeInfo()
VFSVolumeMount()
VFSVolumeSetLabel()
VFSVolumeSize()
VFSVolumeUnmount()

Bluetooth Library Feature Set

Because not every system has (or needs) Bluetooth Library services, applications wishing to use these services should check to make sure they are present before calling them. This is accomplished by checking for the Bluetooth Library's system feature with a call to FtrGet(), supplying `btLibFeatureCreator` for the feature creator and `btLibFeatureVersion` for the feature number.

The following code shows how to check for the presence of the Bluetooth Library.

```
UInt32 btVersion;
```

```
// Make sure Bluetooth components are installed
// This check also ensures Palm OS 4.0 or greater
if (FtrGet(btLibFeatureCreator, btLibFeatureVersion,
    &btVersion) != errNone) {
    // Alert the user if it's the active application
    if ((launchFlags & sysAppLaunchFlagNewGlobals) &&
        (launchFlags & sysAppLaunchFlagUIApp))
        FrmAlert (MissingBtComponentsAlert);
    return sysErrRomIncompatible;
}
```

You can learn more about the Bluetooth Library by reading Chapter 6, "Bluetooth," on page 131 of the *Palm OS Programmer's Companion*, vol. II, *Communications*.

Functions

This feature set adds the following functions.

Bluetooth Library Security Functions

BtLibSecurityFindTrustedDeviceRecord()

BtLibSecurityNumTrustedDeviceRecords()

BtLibSecurityGetTrustedDeviceRecordInfo()

BtLibSecurityRemoveTrustedDevice Record()

Bluetooth Library Utility Functions and Macros

BtLibAddrAToBtd()

BtLibAddrBtdToA()

BtLibL2CapHToNL()

BtLibL2CapHToNS()

BtLibL2CapNToHL()

BtLibL2CapNToHS()

BtLibRfCommHToNL()

BtLibRfCommHToNS()

BtLibRfCommNToHL()

BtLibRfCommNToHS()

BtLibSdpHToNL()

BtLibSdpHToNS()

BtLibSdpNToHL()

BtLibSdpNToHS()

Bluetooth Library Management Functions

BtLibClose()
BtLibOpen()
BtLibCancelInquiry()
BtLibDiscoverMultipleDevices()
BtLibDiscoverSingleDevice()
BtLibGetGeneralPreference()
BtLibGetRemoteDeviceName()
BtLibGetSelectedDevices()
BtLibLinkConnect()
BtLibLinkDisconnect()
BtLibLinkGetState()
BtLibLinkSetState()
BtLibPiconetCreate()
BtLibPiconetDestroy()
BtLibPiconetLockInbound()
BtLibPiconetUnlockInbound()
BtLibRegisterManagementNotification()
BtLibSetGeneralPreference()
BtLibStartInquiry()
BtLibUnregisterManagementNotification()
BtLibManagementCallback()

Bluetooth Library Socket Functions

BtLibSocketAdvanceCredit()
BtLibSocketClose()
BtLibSocketConnect()
BtLibSocketCreate()
BtLibSocketGetInfo()
BtLibSocketListen()
BtLibSocketRespondToConnection()
BtLibSocketSend()

Bluetooth Library Service Discovery Protocol Functions

BtLibSdpCompareUuids()
BtLibSdpGetPSMByUuid()
BtLibSdpGetRawDataElementSize()
BtLibSdpGetRawDataElementType()
BtLibSdpGetServerChannelByUuid()
BtLibSdpParseRawDataElement()
BtLibSdpServiceRecordCreate()
BtLibSdpServiceRecordDestroy()
BtLibSdpServiceRecordGetAttribute()
BtLibSdpServiceRecordGetNumListEntries()
BtLibSdpServiceRecordGetNumLists()
BtLibSdpServiceRecordGetRawAttribute()
BtLibSdpServiceRecordGetSizeOfRawAttribute()
BtLibSdpServiceRecordGetStringOrUrlLength()
BtLibSdpServiceRecordMapRemote()
BtLibSdpServiceRecordSetAttribute()
BtLibSdpServiceRecordSetAttributesForSocket()
BtLibSdpServiceRecordSetRawAttribute()
BtLibSdpServiceRecordsGetByServiceClass()
BtLibSdpServiceRecordStartAdvertising()
BtLibSdpServiceRecordStopAdvertising()
BtLibSdpUuidInitialize()
BtLibSdpVerifyRawDataElement()
BtLibSocketCallback()

High-Density Display Feature Set

You can verify that this feature set is implemented by checking the version of the Window Manager. If the Window Manager version is 4 or greater, the High-Density Display feature set is supported. To check the version of the Window Manager, use this call:

```
err = FtrGet(sysFtrCreator, sysFtrNumWinVersion, &version);
```

Upon return, if `version` has a value of 4 or greater, the High-Density Display feature set is present. Note that just because the High-Density Display feature set is present, it isn't necessarily being used. You may want to check the density of the screen, as follows:

```
WinScreenGetAttribute(winScreenDensity, &attr);
if (attr == kDensityDouble) {
  //the screen is double density
}
```

The 5.0 New Feature Set incorporates all of the functionality present in the High-Density Display Feature Set, so if your application is running on Palm OS Garnet you can assume that the High-Density Display Feature Set is present as well.

New Data Types

This feature set adds the following new data types.

Bitmap Data Types

`BitmapTypeV0`	`BitmapTypeV1`
`BitmapTypeV2`	`BitmapTypeV3`
`DensityType`	`PixelFormatType`

In addition, the definition of `BitmapType` changed.

Font Data Types

`FontDensityType`	`FontTypeV2`

In addition, this feature set defines a new Extended Font Resource.

Window Constants

This feature set adds a new set of Window Coordinate System Constants.

Functions

This feature set adds the following functions.

Bitmap Functions

BmpCreateBitmapV3() BmpGetCompressionType()

BmpGetDensity() BmpGetNextBitmapAnyDensity()

BmpGetTransparentValue() BmpGetVersion()

BmpSetDensity() BmpSetTransparentValue()

System Event Manager Functions

EvtGetPenNative()

Window Functions

WinGetCoordinateSystem() WinGetSupportedDensity()

WinPaintRoundedRectangleFrame() WinPaintTiledBitmap()

WinScaleCoord() WinScalePoint()

WinScaleRectangle() WinScreenGetAttribute()

WinSetCoordinateSystem() WinUnscaleCoord()

WinUnscalePoint() WinUnscaleRectangle()

1.5x Display Feature Set

You can verify that this feature set is implemented by checking the version of the Window Manager. If the Window Manager version is 5 or greater, the 1.5x Display feature set is supported. To check the version of the Window Manager, use this call:

```
err = FtrGet(sysFtrCreator, sysFtrNumWinVersion, &version);
```

Upon return, if `version` has a value of 5 or greater, the 1.5x Display feature set is present. Note that just because the 1.5x Display feature set is present, it isn't necessarily being used. You may want to check the density of the screen, as follows:

```
WinScreenGetAttribute(winScreenDensity, &attr);
if (attr == kDensityOneAndAHalf) {
  //the screen is one-and-a-half density
}
```

New Data Types

This feature set adds the following new data types.

Window Constants

This feature set adds a new set of Scaling Mode Values.

Functions

This feature set adds the following functions.

Window Functions

`WinGetScalingMode()` `WinSetScalingMode()`

Sound Stream Feature Set

The Sound Stream feature set adds a number of "stream" functions and constants to the Sound Manager. You can verify that this feature set is supported by checking for the Sound Manager's version number. If the Sound Manager's version feature is defined, the Sound Stream feature set is supported.

The following code shows how to check for the presence and proper version of the Sound Manger. Note that `expectedSndMgrVersionNum` should be replaced by the actual version number you expect (typically, `sndMgrVersionNum`).

```
UInt32 version;
Err err;

err = FtrGet(sysFileCSoundMgr, sndFtrIDVersion, &version);
```

```
if(err){
    // Sound Stream Feature Set not present
} else {
    // The Sound Stream Feature Set is present.
    // Check version number of Sound Manager,
    // if necessary
    if(version == expectedSndMgrVersionNum)
        // everything is OK
}
```

Sound Stream Data Structures and Types

This feature set adds the following data structures:

SndPtr

SndStreamRef

SndSampleType

SndStreamWidth

SndStreamMode

Sound Stream Enums and Constants

This feature set adds the following enums and constants:

Sound Stream Enums and Constants

SndSampleTag

Stereo Pan Constants

SndStreamModeTag

Volume Constants

SndStreamWidthTag

Sound Resource Playback Flags

Sound Stream Functions

This feature set adds the following functions.

Sound Stream Functions

SndPlayResource()

SndStreamPause()

SndStreamCreate()

SndStreamSetPan()

SndStreamCreateExtended()

SndStreamSetVolume()

SndStreamDelete() SndStreamStart()

SndStreamGetPan() SndStreamStop()

SndStreamGetVolume()

In addition, the Sound Stream Feature Set defines the following callback functions:

SndStreamBufferCallback()

SndStreamVariableBufferCallback()

5.0 New Feature Set

You can check that this feature set is implemented by checking that the system version is 5.0 or higher. Use this FtrGet() call:

```
err = FtrGet(sysFtrCreator,
sysFtrNumROMVersion, &romVersion);
```

The romVersion parameter should be greater than or equal to 0x05003000, which can be constructed using the sysMakeROMVersion macro:

```
sysMakeROMVersion(5,0,0,sysROMStageRelease,0)
```

This feature set corresponds to version 5.0 of Palm OS Garnet.

The Palm OS Application Compatibility Environment (PACE) has its own version associated with it. You can obtain this version number with:

```
err = FtrGet('pace', 0, &paceVersion);
```

Notifications

This feature set adds the following database-related notifications:

sysNotifyDBCreatedEvent sysNotifyDBDirtyEvent
sysNotifyDBChangedEvent

It also broadcasts the following notifications:

sysNotifyAppLaunchingEvent
sysNotifyAppQuittingEvent
sysNotifyEventDequeuedEvent
sysNotifyIdleTimeEvent
sysNotifyInsPtEnableEvent
sysNotifyKeyboardDialogEvent
sysNotifyProcessPenStrokeEvent
sysNotifyVirtualCharHandlingEvent

Functions

This feature set adds the following functions.

ARM-Native Functions

PceNativeCall()

Virtual Ports

Virtual drivers and virtual serial ports are not supported on Palm OS Garnet.

Functions and Traps not Supported by PACE

For various reasons a number of functions and traps are not supported by PACE. The following sections list those functions, grouped according to the reason that they are not implemented.

Unimplemented "System Use Only" Functions

The following functions, which are documented as "System Use Only," are not supported by the Palm OS Application Compatibility Environment (PACE).

AlmAlarmCallback
AlmCancelAll
AlmDisplayAlarm
AlmInit
AlmTimeChange
DmInit
EvtDequeueKeyEvent
EvtGetSysEvent[1]
EvtInitialize
EvtSetKeyQueuePtr
EvtSetPenQueuePtr
EvtSysInit
ExgInit
FrmAddSpaceForObject
FtrInit
GrfFree
GrfInit
InsPtCheckBlink
InsPtInitialize
IntlInit
MemCardFormat
MemHandleFlags
MemHandleOwner
MemHandleResetLock[1]
MemHeapFreeByOwnerID
MemHeapInit
MemInit
MemInitHeapTable
MemKernelInit
MemPtrFlags
MemPtrOwner
MemPtrResetLock
MemStoreSetInfo
PenClose
PenGetRawPen
PenOpen
ScrCompressScanLine
ScrCopyRectangle
ScrDeCompressScanLine
ScrDrawChars
ScrDrawNotify
ScrLineRoutine
ScrRectangleRoutine
ScrScreenInfo
ScrSendUpdateArea
SlkProcessRPC
SlkSysPktDefaultResponse
SndInit
SysBatteryDialog
SysColdBoot
SysDoze
SysInit
SysNewOwnerID
SysSemaphoreSet
SysUILaunch
SysWantEvent
TimInit
UIInitialize
UIReset
WinAddWindow
WinRemoveWindow

1. Implemented on release ROMs but flagged as illegal on debug ROMs.

As a rule, functions and traps that are not documented should be treated as if they are unimplemented and should not be used by applications.

Implemented "System Use Only" Functions and Traps

Although marked "System Use Only," a number of functions and traps are required by applications in the ROM, by Palm Debugger, by test applications, by scripting, or by some popular applications. Because of this, the following System Use Only functions and traps are supported by PACE. Because they are intended only for system use, however, applications should do what they can to avoid using them.

AlmEnableNotification
AttnAllowClose
AttnEnableNotification
AttnIndicatorAllow
AttnIndicatorAllowed
AttnIndicatorGetBlinkPattern
AttnIndicatorSetBlinkPattern
AttnReopen
DmResetRecordStates
EvtEnqueuePenPoint
EvtGetSilkscreenAreaList()
FileReadLow
Find
FrmActiveState
FrmHandleEvent()
HwrDelay
HwrDockStatus
HwrLEDAttributes
HwrMemReadable
HwrMemWritable
HwrVibrateAttributes
MemChunkNew
MemHeapPtr
PenRawToScreen
PenScreenToRaw
SysGetAppInfo
SysLaunchConsole
SndSetDefaultVolume
SysLCDContrast
SysSetA5
SysSetPerformance
SysSleep
SysUIBusy
WinDrawWindowFrame

Obsolete Functions and Traps

The following functions are not supported by PACE because they are obsolete.

FplAdd()[1]
FplAToF()[1]
FplBase10Info()[1]
FplDiv()[1]
FplFloatToLong()[1]
FplFloatToULong()[1]
FplFToA()[1]
FplLongToFloat()[1]
FplMul()[1]
FplSub()[1]
WiCmdV32()

1. Implemented on release ROMs but flagged as illegal on debug ROMs.

'NOP' Functions and Traps

These functions and traps should not be called by applications (many are documented as "System Use Only"). Because some third-party applications do call them, for backward compatibility they act as NOPs.

FplFree()	SerReceiveISP
FplInit()	SrmSleep
HwrEnableDataWrites	SrmWake
HwrDisableDataWrites	SysDisableInts
HwrTimerSleep	SysRestoreStatus
HwrTimerWake	TimHandleInterrupt
KeyResetDoubleTap	TimSleep
KeySleep	TimWake
KeyWake	WinDisableWindow
PenSleep	WinEnableWindow
PenWake	WinInitializeWindow

Unimplemented Rare Functions and Traps

These functions and traps are only used internally by Palm OS, by serial drivers, by OEM extensions, and the like. They are not implemented by PACE.

ConGetS	FlashErase
ConPutS	FlashProgram
DayDrawDays()	IntlSetRoutineAddress()
DayDrawDaySelector()	MemGetRomNVParams
DbgCommSettings	MemNVParams
DbgGetMessage	OEMDispatch[1]
DlkDispatchRequest	ResLoadForm()
DlkStartServer	SlkSetSocketListener()
DmMoveOpenDBContext	SysNotifyDatabaseAdded
DmOpenDBWithLocale	SysNotifyDatabaseRemoved
FlashCompress	SysSetTrapAddress()

1. Supported if the OEM supports the trap.

5.1 New Feature Set

You can check that this feature set is implemented by checking that the system version is 5.1 or higher. Use this FtrGet() call:

```
err = FtrGet(sysFtrCreator,
sysFtrNumROMVersion, &romVersion);
```

The romVersion parameter should be greater than or equal to 0x05103000, which can be constructed using the sysMakeROMVersion macro:

```
sysMakeROMVersion(5,1,0,sysROMStageRelease,0)
```

This feature set corresponds to version 5.1 of Palm OS Garnet.

Net Library Interface Settings

This feature set adds the following Net Library Interface Settings:

DriverVersion	80211AccessPointBSSID
FirmwareVersion	80211AssociationStatus
FirmwareDate	80211MKKCallSign
80211Device	80211CountryTest
80211ESSID	

See NetLibIFSettingGet() for a description of each of the above settings.

CPM Library

This feature set includes the Cryptographic Provider Manager (CPM) shared library. The CPM library isn't automatically loaded upon system boot: before making use of the CPM library you must first load it, using SysLibFind and SysLibLoad.

The CPM library defines a number of constants:

- AP Capability Constants
- Block Encryption Mode Constants
- Cipher Algorithm Constants
- Export Encoding Constants

- Hashing Algorithm Constants
- Key Class Constants
- Key Usage Constants
- Plaintext Padding Constants

The following structures are defined as part of the CPM library:

APCipherInfoStruct
APHashInfoStruct
APKeyInfoStruct
APProviderContextStruct
APProviderInfoStruct
APVerifyInfoStruct
CPMInfoStruct

Finally, the CPM library contains the following functions:

CPMLibDecrypt()
CPMLibDecryptFinal()
CPMLibDecryptInit()
CPMLibDecryptUpdate()
CPMLibEncrypt()
CPMLibEncryptFinal()
CPMLibEncryptInit()
CPMLibEncryptUpdate()
CPMLibExportCipherInfo()
CPMLibExportHashInfo()
CPMLibExportKeyInfo()
CPMLibExportVerifyInfo()
CPMLibGenerateKey()
CPMLibGetInfo()
CPMLibGetProviderInfo()
CPMLibHash()
CPMLibHashFinal()
CPMLibHashInit()
CPMLibImportCipherInfo()
CPMLibImportHashInfo()
CPMLibImportKeyInfo()
CPMLibImportVerifyInfo()
CPMLibReleaseCipherInfo()
CPMLibReleaseHashInfo()
CPMLibReleaseKeyInfo()
CPMLibReleaseVerifyInfo()
CPMLibVerify()
CPMLibVerifyFinal()
CPMLibVerifyInit()
CPMLibVerifyUpdate()

SSL Library

The 5.1 New Feature Set includes the Secure Sockets Layer (SSL) shared library. The SSL library isn't automatically loaded upon system boot: before making use of the SSL library you must first load it and open it, using code similar to the following:

```
Err error;
UInt16 libRef;

if ( SyLibFind( kSslDBName, &libRef ) != 0 )
{
  error = SysLibLoad(kSslLibType, kSslLibCreator, &libRef);
}
/* error checking goes here. */

error = SslLibOpen( libRef );
...
```

SSL Library Structures and Data Types

The SSL library defines a number of structures and data types:

SslAttribute	SslIoBuf
SslCallback	SslLib
SslCipherSuiteInfo	SslSession
SslContext	SslSocket

SSL Library Functions

SslClose()	SslLibGetPtr()
SslConsume()	SslLibName
SslContextCreate()	SslLibOpen()
SslContextDestroy()	SslLibSetLong()
SslContextGetLong()	SslLibSetPtr()
SslContextGetPtr()	SslLibSleep
SslContextSetLong()	SslLibWake
SslContextSetPtr()	SslOpen()
SslFlush()	SslPeek()
SslLibClose()	SslRead()
SslLibCreate()	SslReceive()
SslLibDestroy()	SslSend()
SslLibGetLong()	SslWrite()

In addtion, the SSL library defines one callback function:

- SslCallbackFunc()

SSL Library Attributes and Macros

You interact with SSL attributes using the macros declared in `SslLibMac.h`. The attributes themselves are documented in Chapter 8, "Secure Sockets Layer (SSL)," on page 203 of *Palm OS Programmer's Companion*, vol. II, *Communications*.

Pen Input Manager Feature Set

To see if a device contains a dynamic input area controlled by the Pen Input Manager, check the `pinFtrAPIVersion` feature as shown in Listing 89.1

Listing 89.1 Checking the dynamic input area feature

```
err = FtrGet(pinCreator, pinFtrAPIVersion, &version);
if (!err && version) {
```

```
   //PINS exists
}
```

Pen Input Manager Version 1.0

If the value of the Pen Input Manager feature is `pinAPIVersion1_0`, the following functions are defined:

`FrmGetDIAPolicyAttr`

`FrmSetDIAPolicyAttr`

`PINGetInputAreaState`

`PINGetInputTriggerState`

`PINSetInputAreaState`

`PINSetInputTriggerState`

`WinSetConstraintsSize`

And the following notification is defined:

`sysNotifyDisplayResizedEvent`

Pen Input Manager Version 1.0

If the value of the Pen Input Manager feature is `pinAPIVersion1_1`, the following functions are defined in addition to the ones listed in the previous section:

`StatGetAttribute`

`StatHide`

`StatShow`

`SysGetOrientation`

`SysGetOrientationTriggerState`

`SysSetOrientation`

`SysSetOrientationTriggerState`

and the `winDisplayChangedEvent` is defined as well.

Graffiti 2 Feature Set

Version 4.1.2 of Palm OS and version 5.2 of Palm OS Garnet use the Graffiti® 2 handwriting recognition engine, which is based on Jot® technology. The Graffiti Manager still provides the interface to the new engine; however, some functions have been deprecated.

You can check that this feature set is implemented by checking for the existence of the Graffiti 2 dynamic ID, which is set by the Graffiti 2 library. Use code similar to the following:

```
if (!FtrGet('grft', 1110, &g2DynamicID) ||
  !FtrGet('grf2', 1110, &g2DynamicID)) {
    // Graffiti 2 feature set is present.
}
```

Note that the Graffiti 2 creator code changed between version 4.1.2 and version 5.2, so both `FtrGet()` calls are necessary.

Deprecated Functions

The following functions are deprecated if this feature set is present:

`GrfAddPoint()`

`GrfFlushPoints()`

`GrfInitState()` (this was reinstated in later versions)

`GrfCleanState()`

`GrfMatch()`

`GrfFilterPoints()`

`GrfGetNumPoints()`

`GrfGetPoint()`

`GrfFindBranch()`

`GrfMatchGlyph()`

`GrfGetGlyphMapping()`

In addition, the `capsLock` and `numLock` parameters to `GrfSetState()` and `GrfGetState()` do not apply for the Graffiti 2 recognizer.

Note that the Graffiti Shift Indicator APIs work as normal.

5.3SC New Feature Set

You can check that this feature set is implemented by checking that the system version is 5.3 or higher. Use this `FtrGet()` call:

```
err = FtrGet(sysFtrCreator, sysFtrNumROMVersion,
   &romVersion);
```

The `romVersion` parameter should be greater than or equal to `0x05303000`, which can be constructed using the `sysMakeROMVersion` macro:

```
sysMakeROMVersion(5,3,0,sysROMStageRelease,0)
```

This feature set corresponds to version 5.3SC of Palm OS Garnet.

Functions

This feature set adds the following functions.

Glue Functions

`LmGlueGetSystemLocale()`

Notifications

This feature set adds the following notifications.

`sysNotifyGsiDrawIndicator`

`sysNotifyInputAreaDrawingEvent`

`sysNotifyInputAreaPendownEvent`

`sysNotifySelectDay`

Existing Functions that Changed

The following functions that existed prior to 5.3SC have changed in release 5.3SC:

The `capsLock` and `numLock` parameters to `GrfSetState()` and `GrfGetState()` now apply to the Graffiti 2 recognizer. In the Palm OS Garnet operating system, Graffiti 2 was introduced in version 5.2. In that initial implementation, these two parameters did not apply.

In version 5.2, `GrfInitState()` was deprecated and nonfunctional. This function is now supported again in version 5.3SC. (Actually, it was reinstated in version 5.2.1 and later.)

In the following functions, the `numOfElements` parameter was changed to a `UInt16` type: `SysBinarySearch`, `SysInsertionSort`, and `SysQSort`.

5.4 New Feature Set

You can check that this feature set is implemented by checking that the system version is 5. 4 or higher. Use this `FtrGet()` call:

```
err= FtrGet(sysFtrCreator, sysFtrNumROMVersion, &romVersion);
```

The `romVersion` parameter should be greater than or equal to `0x05403000`, which can be constructed using the `sysMakeROMVersion` macro:

```
sysMakeROMVersion(5,4,0,sysROMStageRelease,0)
```

This feature set corresponds to version 5.4 of Palm OS Garnet.

New Data Types, Enums, Constants

The following data types, enums, and constants are new. They support socket notifications:

- `NetSocketNoticeType`
- `SysNotifyNetSocketType`
  See "Socket Notification Specific Data" on page 87
- `NoticeTypeEnum`
- netSocketNotice. . . constants
  See "Socket Notice Trigger Conditions" on page 1468.

For details on socket notices and socket notifications, see Chapter 7, "Socket Notices," on page 188, in *Palm OS Programmer's API Reference*.

Existing Functions that Changed

The following function that existed prior to 5.4 has changed in release 5.4:

The *optionValueP* parameter of `NetLibSocketOptionSet()` is now of type `NetSocketNoticeType` when you are also passing in parameters *netSocketOptLevelSocket* and *netSocketOptSocketNotice*. This call is made when you are registering to receive socket notifications.

Other Changes

Two former Palm OS limits are no longer dictated by PalmSource, Inc: the maximum number of simultaneous active sockets and the maximum size of the stack available to Net Library. Licensees now decide on the number of sockets and the stack size they deem appropriate.

Bluetooth Sleep and Wake Scheduling Feature Set

You can check that this feature set is implemented by checking for the `sysFtrBtSupportsScheduledWakeup` feature. Use this `FtrGet()` call:

```
err = FtrGet(sysFtrCreator,
sysFtrBtSupportsScheduledWakeup, &value);
```

If Bluetooth sleep and wake scheduling is available, the `value` parameter will be non-zero and the returned error should also be zero (for no error).

For more information about sleep and wake scheduling, see Chapter 6, "Radio Power Management," on page 140.

ARM Debug Nub Feature Set

You can check that this feature set is implemented by checking for the `adnFtrCreator` feature. Use this `FtrGet()` call:

```
err = FtrGet(adnFtrCreator, adnFtrNumVersion, &value);
```

If the ARM debug nub feature set is present, the returned error should be `errNone`.

ARM Debug Nub Functions and Macros

`AdnDebugBreak()`	`AdnDebugEnableGet()`
`AdnDebugEnableGetSupported()`	`AdnDebugEnableSet()`
`AdnDebugLicenseeSpecific()`	`AdnDebugMessage()`
`AdnDebugMessageIf()`	`AdnDebugNativeRegister()`
`AdnDebugNativeRegisterAddr()`	`AdnDebugNativeUnregister()`
`AdnDebugNativeUnregisterAddr()`	`AdnDebugUpdateLoadedModules()`
`AdnGetNativeCodeBaseAddr()`	`SetR10ToDataPointer()`

Palm OS Cobalt Feature Set

You can check that this feature set is implemented by checking that the system version is 6.0 or higher. Use this `FtrGet()` call:

```
err= FtrGet(sysFtrCreator, sysFtrNumROMVersion, &romVersion);
```

The `romVersion` parameter should be greater than or equal to `0x06003000`, which can be constructed using the `sysMakeROMVersion` macro:

```
sysMakeROMVersion(6,0,0,sysROMStageRelease,0)
```

This feature set corresponds to version 6.0.1 of Palm OS Cobalt.

PACE

The Palm Application Compatibility Environment, or PACE, is a 68K emulator that supports execution of well-behaved 68K-based Palm OS applications on devices that use an ARM processor.

PACE allows the majority of existing 68K-based Palm OS applications to run on devices that use an ARM processor. Users can beam an application from a 68K device to an ARM-based device and run the application. The Palm Application Compatibility Environment helps provide a migration path for developers. The developer can continue to use existing 68K-based tools to build their application.

NOTE: Both Palm OS Garnet and Palm OS Cobalt use PACE to provide 68K emulation. PACE relies on the underlying operating system to provide much of its functionality. Because Palm OS Garnet is architecturally very similar to 68K-native versions of Palm OS, the version of PACE running on Palm OS Garnet provides a high degree of compatibility with a Palm Powered device containing a 68K-family processor. Palm OS Cobalt, however, is architecturally very different than all earlier Palm OS versions. Reflecting this, PACE on Palm OS Cobalt is not as compatible as PACE on Palm OS Garnet. So that your 68K applications will run on the widest range of Palm Powered devices, take careful note of the following sections which describe the behavior of PACE on Palm OS Cobalt.

Design Objectives

The Palm Application Compatibility Environment is designed to allow well-behaved 68K applications to run at 68K speeds or faster on an ARM-based device, with minimum code and memory overhead. A well-behaved application is one that:

- only uses documented Palm OS APIs
- does not access hardware directly
- does not access the display memory directly
- does not access low memory globals
- does not access the fields of Palm OS structures directly

- runs on Palm OS Emulator with a debug Palm OS Garnet ROM without encountering any errors

Performance

Performance of a 68K application varies greatly depending on how much time it spends executing 68K instructions compared to the time it spends calling Palm OS functions.

Code that consists only of 68K instructions, such as a prime number generator, will run slower than an ARM-native version of the same code since the 68K instructions are emulated. For reference, the time it takes to execute emulated 68K instructions on a 70 MHz ARM device is roughly the same as the time it takes to execute the same instructions on a Palm Vx device. Most applications spend a great deal of time inside operating system calls, however, and those calls execute at the full speed of the ARM processor (note that there may be additional overhead for some Palm OS functions, depending on how close the native function's API is to the 68K API). Thus most 68K applications running under PACE will actually run much faster than they would on a device with a 68K processor.

Developer SDK

Because an application that runs under PACE is like any other 68K application, when writing applications to run under PACE you continue to use the tools and headers you would use to create a 68K-based Palm OS application.

Applications are no longer allowed access to many internal, publicly-defined structures (such as the `ControlType` structure). To make up for this, some accessor functions were added in Palm OS 4.0, and additional accessor functions were added to the Palm OS glue library shipped with the Palm OS Garnet SDK.

The 68K Palm debugger and other tools which depend on the 68K debugger APIs (such as the Metrowerks debugger) are supported by the Palm Application Compatibility Environment.

API Restrictions

Most well-behaved applications run under the Palm OS Garnet version of PACE with no problems. Due to differences in the underlying operating system in Palm OS Cobalt, however, PACE is

somewhat more restricting on Palm OS Cobalt devices. The following sections detail those restrictions.

Deprecated APIs

Very few deprecated APIs (such as `CategoryEditV20()`) are supported for native ARM applications. The Palm Application Compatibility Environment still supports these deprecated APIs, unless they are listed in Table 89.2 on page 2445.

Unsupported APIs

A number of Palm OS APIs are not supported by PACE. These are APIs that either are documented as private, are internal-only APIs (yet appear in public header files), or are APIs that developer support has advised developers to not use. A list of unsupported APIs can be found under "Unsupported Palm OS Functions" on page 2444.

Card Number Argument

The native ARM version of Palm OS no longer has the concept of memory cards. For this reason, the card number concept is faked for emulated 68K applications. If an application calls `MemNumCards()`, a value of 1 is always returned. If an application calls any function that takes *`cardNum`* as an argument and the value for *`cardNum`* is not zero, an error is returned to the application.

Record Unique IDs

In previous versions of Palm OS, only 24 bits of a record's unique ID were stored in the record header. In Palm OS Cobalt, however, all 32 bits are unique. The function that returns a record's unique ID returns a 32-bit value; to ensure the greatest degree of compatiblity an application should save all 32 bits of the record's unique ID, and not truncate the result to 24 bits.

Effect of Calling an Unsupported or Deprecated Palm OS Function

If a 68K application calls an unsupported or deprecated Palm OS function, an alert is displayed and the application is terminated. The alert contains the message "An error occurred in the application you are using. Note the error code and contact the developer of this

application" followed by an error number in parentheses. When the user presses the OK button, the application is forced to exit. The debug version of this error alert has two more numbers displayed (to help you pinpoint the problem) and a Cancel button. If you tap the Cancel button, PACE tries to connect to the 68K Palm Debugger so you can determine why and where the error is occurring.

Unsupported Palm OS Functions

Table 89.2 is a list of Palm OS functions that are not supported by PACE in Palm OS Cobalt. The following are the reasons why these functions are not supported.

Documented as "System Use Only": These functions are documented as "System Use Only" in the *Palm OS Programmer's API Reference* and thus should never have been called by applications.

Should have been documented as "system use only": These functions were intended for internal PalmSource use only but were documented.

Obsolete: These functions are not implemented because they have long been obsolete. Current Palm OS applications should no longer be using them.

Implemented as a "NOP" function: Typically, these are functions that should not have been called by applications. Because some applications may call them, however, PACE supports them. However, they do nothing and simply return.

Rarely-used function: Functions that are only used internally by Palm OS, by serial drivers, or by OEM extensions. They are not functions that an application would use. PACE does not implement these functions.

Unimplemented in Palm OS Cobalt: Functions that were supported by PACE in Palm OS Garnet but are no longer supported by PACE in Palm OS Cobalt. The vast majority of these are intended for system use only, and the remaining few are very rarely used by applications.

Table 89.2 Unsupported Palm OS functions

Function	Unsupported because...
`AlmAlarmCallback()`	Documented as "System Use Only"
`AlmCancelAll()`	Documented as "System Use Only"
`AlmDisplayAlarm()`	Documented as "System Use Only"
`AlmTimeChange()`	Documented as "System Use Only"
`ConGetS()`	Rarely-used function
`ConPutS()`	Rarely-used function
`DayDrawDays()`	Rarely-used function
`DayDrawDaySelector()`	Rarely-used function
`DayHandleEvent()`	Should have been documented as "System Use Only"
`DbgCommSettings()`	Rarely-used function
`DbgGetMessage()`	Rarely-used function
`DlkControl()`	Rarely-used function
`DlkDispatchRequest()`	Rarely-used function
`DlkStartServer()`	Rarely-used function
`DmInit()`	Documented as "System Use Only"
`EvtDequeueKeyEvent()`	Documented as "System Use Only"
`EvtEnqueuePenPoint()`	Was documented as "System Use Only." In Palm OS Cobalt, returns `sysErrNotAllowed`. On debug ROMs, displays a fatal alert.
`EvtGetPenBtnList()`	Rarely-used function
`EvtGetSilkscreenAreaList()`	Rarely-used function
`EvtGetSysEvent()`	Documented as "System Use Only"

Table 89.2 Unsupported Palm OS functions *(continued)*

Function	Unsupported because...
EvtKeyQueueSize()	In Palm OS Cobalt, returns sysErrNotAllowed. On debug ROMs, displays a fatal alert.
EvtPenQueueSize()	In Palm OS Cobalt, returns sysErrNotAllowed. On debug ROMs, displays a fatal alert.
EvtProcessSoftKeyStroke()	In Palm OS Cobalt, returns sysErrNotAllowed. On debug ROMs, displays a fatal alert.
EvtSetKeyQueuePtr()	Documented as “System Use Only”
EvtSetPenQueuePtr()	Documented as “System Use Only”
EvtSysInit()	Documented as “System Use Only”
ExpCardGetSerialPort()	Unimplemented in Palm OS Cobalt
ExpCardInserted()	Documented as “System Use Only”
ExpCardRemoved()	Documented as “System Use Only”
ExpSlotDriverInstall()	Unimplemented in Palm OS Cobalt
ExpSlotDriverRemove()	Unimplemented in Palm OS Cobalt
ExpSlotLibFind()	Unimplemented in Palm OS Cobalt
ExpSlotRegister()	Documented as “System Use Only”
ExpSlotUnregister()	Documented as “System Use Only”
FplAdd()	Obsolete
FplAToF()	Obsolete
FplBase10Info()	Obsolete
FplDiv()	Obsolete
FplFloatToLong()	Obsolete
FplFloatToULong()	Obsolete

Table 89.2 Unsupported Palm OS functions *(continued)*

Function	Unsupported because...
FplFree()	Implemented as a "NOP" function
FplFToA()	Obsolete
FplInit()	Implemented as a "NOP" function
FplLongToFloat()	Obsolete
FplMul()	Obsolete
FplSub()	Obsolete
FrmAddSpaceForObject()	Documented as "System Use Only"
FtrInit()	Documented as "System Use Only"
GrfFree()	Documented as "System Use Only"
GrfInit()	Documented as "System Use Only"
InsPtCheckBlink()	Documented as "System Use Only"
InsPtEnable()	Implemented as a "NOP" function
InsPtEnabled()	Implemented as a "NOP" function
InsPtGetHeight()	Implemented as a "NOP" function
InsPtGetLocation()	Implemented as a "NOP" function
InsPtInitialize()	Documented as "System Use Only"
InsPtSetHeight()	Implemented as a "NOP" function
InsPtSetLocation()	Implemented as a "NOP" function
KbdDraw()	Documented as "System Use Only"
KbdErase()	Documented as "System Use Only"
KbdGetLayout()	Documented as "System Use Only"
KbdGetPosition()	Documented as "System Use Only"
KbdGetShiftState()	Documented as "System Use Only"

Table 89.2 Unsupported Palm OS functions *(continued)*

Function	Unsupported because...
`KbdHandleEvent()`	Documented as “System Use Only”
`KbdSetLayout()`	Documented as “System Use Only”
`KbdSetPosition()`	Documented as “System Use Only”
`KbdSetShiftState()`	Documented as “System Use Only”
`KeyboardStatusFree()`	Documented as “System Use Only”
`KeyboardStatusNew()`	Documented as “System Use Only”
`MemCardFormat()`	Documented as “System Use Only”
`MemHandleFlags()`	Documented as “System Use Only”
`MemHandleOwner()`	Documented as “System Use Only”
`MemHandleResetLock()`	Documented as “System Use Only”
`MemHeapFreeByOwnerID()`	Documented as “System Use Only”
`MemHeapInit()`	Documented as “System Use Only”
`MemInit()`	Documented as “System Use Only”
`MemInitHeapTable()`	Documented as “System Use Only”
`MemKernelInit()`	Documented as “System Use Only”
`MemPtrFlags()`	Documented as “System Use Only”
`MemPtrOwner()`	Documented as “System Use Only”
`MemPtrResetLock()`	Documented as “System Use Only”
`MemSemaphoreRelease()`	Documented as “System Use Only”
`MemSemaphoreReserve()`	Documented as “System Use Only”
`MemStoreSetInfo()`	Documented as “System Use Only”
`OmGetIndexedLocale()`	Unimplemented in Palm OS Cobalt
`OmGetNextSystemLocale()`	Unimplemented in Palm OS Cobalt

Table 89.2 Unsupported Palm OS functions *(continued)*

Function	Unsupported because...
OmSetSystemLocale()	Unimplemented in Palm OS Cobalt
PenCalibrate()	Implemented as a "NOP" function
PenClose()	Documented as "System Use Only"
PenGetRawPen()	Documented as "System Use Only"
PenOpen()	Documented as "System Use Only"
PenRawToScreen()	Was documented as "System Use Only." In Palm OS Cobalt, returns sysErrNotAllowed. On debug ROMs, displays a fatal alert.
PenResetCalibration()	Implemented as a "NOP" function
PenScreenToRaw()	Was documented as "System Use Only." In Palm OS Cobalt, returns sysErrNotAllowed. On debug ROMs, displays a fatal alert.
PenSleep()	Was documented as "System Use Only." In Palm OS Cobalt this is implemented as a "NOP" function.
PenWake()	Was documented as "System Use Only." In Palm OS Cobalt this is implemented as a "NOP" function.
ResLoadForm()	Rarely-used function
SerReceiveISP()	Implemented as a "NOP" function
SlkProcessRPC()	Documented as "System Use Only"
SlkSetSocketListener()	Rarely-used function
SlkSysPktDefaultResponse()	Documented as "System Use Only"
SndInit()	Documented as "System Use Only"
SndInterruptSmfIrregardless()	Rarely-used function

Table 89.2 Unsupported Palm OS functions *(continued)*

Function	Unsupported because...
SndPlaySmfIrregardless()	Rarely-used function
SndPlaySmfResourceIrregardless()	Rarely-used function
SrmOpenBackground()	Rarely-used function
SrmSleep()	Implemented as a “NOP” function
SrmWake()	Implemented as a “NOP” function
SysBatteryDialog()	Documented as “System Use Only”
SysColdBoot()	Documented as “System Use Only”
SysDisableInts()	Was documented as “System Use Only.” In Palm OS Cobalt this is implemented as a “NOP” function.
SysDoze()	Documented as “System Use Only”
SysFatalAlertInit()	Undocumented “system use only”function.
SysInit()	Documented as “System Use Only”
SysLaunchConsole()	Documented as “System Use Only”
SysNewOwnerID()	Documented as “System Use Only”
SysNotifyBroadcastFromInterrupt()	Rarely-used function
SysRestoreStatus()	Was documented as “System Use Only.” In Palm OS Cobalt this is implemented as a “NOP” function.
SysSemaphoreSet()	Documented as “System Use Only”
SysSetTrapAddress()	Rarely-used function
SysUILaunch()	Documented as “System Use Only”
SysUnimplemented()	Documented as “System Use Only”
TimInit()	Documented as “System Use Only”

Table 89.2 Unsupported Palm OS functions *(continued)*

Function	Unsupported because...
VFSInstallFSLib()	Unimplemented in Palm OS Cobalt
VFSRemoveFSLib()	Unimplemented in Palm OS Cobalt
WinAddWindow()	Documented as "System Use Only"
WinDisableWindow()	Was documented as "System Use Only." In Palm OS Cobalt this is implemented as a "NOP" function.
WinEnableWindow()	Was documented as "System Use Only." In Palm OS Cobalt this is implemented as a "NOP" function.
WinGetFirstWindow()	Rarely-used function
WinInitializeWindow()	Was documented as "System Use Only." In Palm OS Cobalt this is implemented as a "NOP" function.
WinMoveWindowAddr()	Documented as "System Use Only"
WinRemoveWindow()	Documented as "System Use Only"
WinScreenInit()	Documented as "System Use Only"

Accessing the PIM Application Databases

Some 68K applications access the PIM application databases. In Palm OS Cobalt the PIM applications are ARM-native applications, and those applications now use schema databases to hold application data. Although the "classic" PIM application databases are no longer present, PACE does what it can to "do the right thing" when it detects that a 68K application is attempting to access a PIM application database.

NOTE: The PIM database access solution described here depends entirely on the presence of the ARM-native PIM applications. As well, those applications must correctly support the `sysAppLaunchCmdImportRecord`, `sysAppLaunchCmdExportRecord` and `sysAppLaunchCmdDeleteRecord` launch commands. PACE further assumes that the database names, database schema names, and field IDs haven't changed (additional fields may be present, however, and the field order can be changed).

When an application tries to access a 68K PIM application database—either to open or to find it—PACE creates the 68K database, opens it, initializes its application info block (using category names from the corresponding native PIM app) and creates a cache of all records in the corresponding ARM-native PIM application database. Each cache entry contains the following information:

- the unique ID of the native record
- handle to the 68K record
- index of the 68K record
- category that the record belongs to
- attribute flags (dirty and deleted)

Each entry is initialized with the unique ID and category information. All other fields are cleared.

When the application then tries to read a record (for example, by calling `DmQueryRecord()` or `DmGetRecord()`), PACE sends a `sysAppLaunchCmdExportRecord` request to the native application. In response, the native application returns the record in vCard format. PACE converts the vCard information into a 68K PIM application record and stores that record into the 68K PIM application database that was created when the database was opened. The index and record handle of this record are stored in the cache; any further requests for the record will return the record handle from the cache.

If the application creates a new record, PACE creates a record in the 68K PIM application database, adds an entry to the cache, and initializes the cache entry's unique ID to `NULL`.

When the application writes to the new record, PACE marks the record in the cache as "dirty" so that it will be written back to the ARM-native PIM application database when the database is flushed. At that time, PACE converts the 68K record to its vCard representation and then sends a `sysAppLaunchCmdImportRecord` request to the native application. Records are flushed:

- when a database is closed
- whenever another record is created
- before a unique ID is returned to an application (in response to a call to `DmGetRecordInfo()`).

When an application deletes a PIM database record, PACE marks the record in the cache as "deleted" and sends a `sysAppLaunchCmdDeleteRecord` request to the native application.

Once the 68K application closes the PIM application database, PACE flushes any dirty records that have yet to be written, frees up memory allocated for the cache, and closes and deletes the 68K PIM application database.

The two functions `DmFindDatabase()` and `DmGetNextDatabaseByTypeCreator()` have the side effect of leaving the 68K PIM application database around. To clean up these stray databases, PACE deletes any existing 68K PIM application databases whenever a 68K application exits. Note that if you call `DmOpenDatabase()` after locating a database with either of these two functions, and later call `DmCloseDatabase()`, the database will be deleted when the close function is called.

Although `DmFindDatabase()` leaves the 68K PIM application database around, other Data Manager calls may cause PACE to delete it. Because of this, if you are going to open a PIM application database by calling `DmFindDatabase()` followed by `DmOpenDatabase()` (as opposed to the more common method of using `DmOpenDatabaseByTypeCreator()`), avoid making other

Data Manager calls on the PIM application database between the call to `DmFindDatabase()` and `DmOpenDatabase()`.

Limitations

The PIM application database access solution provided by PACE has a number of limitations. For one, not all database access functions are supported. Here is a list of the known limitations:

- It can be slow. The ARM-native PIM applications are sub-launched for each PIM database read, write, or delete.
- Category changes are not reflected back to the native PIM application. Category information should be treated as read-only.
- All three "varieties" of record delete—`DmDeleteRecord()`, `DmRemoveRecord()` and `DmArchiveRecord()`—become requests for the native PIM application to delete the record.
- Some things in the application info blocks are "hard-coded." For example, the ToDo sort order is always priority/due date and the Address Book labels are hard-coded to English text labels.
- The `totalBytes` and `dataBytes` sizes returned from `DmDatabaseSize()` come from the 68K database and thus are usually wrong. They would only be correct if every record has been read in from the ARM-native PIM application database.
- Some database record access functions are not supported, including `DmRemoveSecretRecords()`, `DmMoveCategory()` and `DmDeleteCategory()`.
- There are some limitations with private records. Private records will be masked or hidden when a 68K PIM application is run, but changing the "maskedness" or "hiddenness" has no effect while running the 68K PIM application.
- Because records are not immediately written to the ARM-native PIM application databases, some data may be lost while running a 68K PIM application replacement if the system is reset while the application is running.

Summary of PIM Database Access APIs

Table 89.3 lists those functions for which PACE checks to see if a PIM application database is being accessed. If so, PACE acts as described in the second column.

Table 89.3 Functions that PACE monitors for PIM database access

Function	How treated by PACE
`CategoryCreateList()`	Calls through to the native OS.
`CategoryEdit()`	Calls through to the native OS.
`CategoryFind()`	Calls through to the native OS.
`CategoryFreeList()`	Calls through to the native OS.
`CategoryGetName()`	Calls through to the native OS.
`CategoryGetNext()`	Calls through to the native OS.
`CategorySelect()`	Calls through to the native OS.
`CategorySetName()`	Calls through to the native OS.
`DmArchiveRecord()`	PACE handles calls to this function.
`DmAttachRecord()`	PACE handles calls to this function.
`DmCloseDatabase()`	PACE handles calls to this function.
`DmDatabaseInfo()`	Calls through to the native OS.
`DmDatabaseProtect()`	Calls through to the native OS. Note that applications are not expected to call this function with a PIM database.
`DmDatabaseSize()`	PACE handles calls to this function.
`DmDeleteCategory()`	Not supported when accessing a PIM database. `sysErrNotAllowed` is returned.
`DmDeleteDatabase()`	Calls through to the native OS. Note that applications are not expected to call this function with a PIM database.

Table 89.3 Functions that PACE monitors for PIM database access *(continued)*

Function	How treated by PACE *(continued)*
DmDeleteRecord()	PACE handles calls to this function.
DmDetachRecord()	PACE handles calls to this function.
DmFindDatabase()	PACE handles calls to this function.
DmFindRecordByID()	PACE handles calls to this function.
DmFindRecordByOffsetInCategory()	PACE handles calls to this function.
DmFindSortPosition()	PACE handles calls to this function.
DmGetAppInfoID()	Calls through to the native OS.
DmGetDatabaseLockState()	Calls through to the native OS. Note that applications rarely call this function with a PIM database.
DmGetNextDatabaseByTypeCreator()	PACE handles calls to this function.
DmGetRecord()	PACE handles calls to this function.
DmInsertionSort()	PACE handles calls to this function.
DmMoveCategory()	Not supported when accessing a PIM database. `sysErrNotAllowed` is returned.
DmMoveRecord()	PACE handles calls to this function.
DmNewHandle()	PACE handles calls to this function.
DmNewRecord()	PACE handles calls to this function.
DmNextOpenDatabase()	Calls through to the native OS. Note that applications rarely call this function with a PIM database.
DmNumDatabases()	PACE handles calls to this function.
DmNumRecordsInCategory()	PACE handles calls to this function.
DmOpenDatabase()	PACE handles calls to this function.

Table 89.3 Functions that PACE monitors for PIM database access *(continued)*

Function	How treated by PACE *(continued)*
DmOpenDatabaseByTypeCreator()	PACE handles calls to this function.
DmOpenDatabaseInfo()	Calls through to the native OS.
DmOpenDBNoOverlay()	PACE handles calls to this function.
DmPositionInCategory()	PACE handles calls to this function.
DmQueryNextInCategory()	PACE handles calls to this function.
DmQueryRecord()	PACE handles calls to this function.
DmQuickSort()	PACE handles calls to this function.
DmRecordInfo()	PACE handles calls to this function.
DmReleaseRecord()	PACE handles calls to this function.
DmRemoveRecord()	PACE handles calls to this function.
DmRemoveSecretRecords()	Not supported when accessing a PIM database. `sysErrNotAllowed` is returned.
DmResetRecordStates()	(System Use Only function) Calls through to the native OS. Note that applications rarely call this function with a PIM database.
DmResizeRecord()	PACE handles calls to this function.
DmSetDatabaseInfo()	Calls through to the native OS.
DmSetRecordInfo()	PACE handles calls to this function.

New Functions and Macros

The Palm OS Cobalt feature set adds the following functions.

Certificate Manager Functions

CertMgrAddCert()
CertMgrExportCert()
CertMgrFindCert()
CertMgrGetField()
CertMgrImportCert()
CertMgrReleaseCertInfo()
CertMgrRemoveCert()
CertMgrVerifyCert()

Certificate Manager Macros

apCertMgrElementFieldRDNOIDN()

apCertMgrElementFieldRDNValueN()

apCertMgrElementFieldX509ExBytesN()

apCertMgrElementFieldX509ExCriticalN()

apCertMgrElementFieldX509ExOIDN()

CertMgrVerifyFailure()

New Data Structures and Constants

The following data structures are new. They support the Certificate Manager:

Certificate Manager Data Structures

CertMgrCertChainType
CertMgrCertElementEnum
CertMgrCertFieldEnum
CertMgrCertInfoType
CertMgrCertSearchEnum
CertMgrElementListType
CertMgrElementType
CertMgrVerifyResultType

A number of constants are also defined in the header file that declares the Certificate Manager APIs. They are organized into the following categories:

- X509Cert Element Fields
- RSA Element Fields
- RDN Element Fields

- X509Extensions Element Fields
- Data Types
- Certificate Formats
- Certificate Manager Error Codes
- Certificate Verification Failure Codes

Existing Functions that Changed

The coordinate scaling algorithm changed slightly for the following functions:

Window Manager Functions

`WinScaleCoord()`	`WinScalePoint()`
`WinUnscaleCoord()`	`WinUnscalePoint()`

One-Handed Navigation Feature Set

You can check that this feature set is implemented by checking for the presence of the the `sysFtrNumFiveWayNavVersion` feature. Use this `FtrGet()` call:

```
err = FtrGet(sysFtrCreator, sysFtrNumFiveWayNavVersion, &value);
if (err == errNone){
   // One-handed navigation is supported
} else {
   // No support for one-handed navigation
}
```

On Handspring devices from palmOne (such as the Treo 600), you use a similar technique but check for the `hsFtrIDNavigationSupported` instead. For convenience, the Palm OS Glue library—which contains "glue" equivalents of a number of one-handed navigation functions that will run on devices where either feature is present—contains a function that will check for the presence of either feature. If `FrmGlueNavIsSupported()` returns `true`, one or the other features is present on the device. Note, however, that you should use the Palm OS Glue versions of the `FrmNav...` functions since

the functions declared in `Form.h` only work on devices where the `sysFtrNumFiveWayNavVersion` feature is present.

New Functions and Macros

The One-Handed Navigation feature set adds the following functions.

Form Functions

FrmCountObjectsInNavOrder()	FrmGetNavEntry()
FrmGetNavOrder()	FrmGetNavState()
FrmNavDrawFocusRing()	FrmNavGetFocusRingInfo()
FrmNavObjectTakeFocus()	FrmNavRemoveFocusRing()
FrmSetNavEntry()	FrmSetNavOrder()
FrmSetNavState()	

New Data Structures and Types

The following data structures and types are new. They support the one-handed navigation feature:

Form Data Structures and Types

FrmNavHeaderType	FrmNavOrderEntryType
FrmNavFocusRingStyleEnum	FrmNavHeaderFlagsType
FrmNavObjectFlagsType	FrmNavStateFlagsType

In addition this feature set contains a number of constants specific to the one-handed navigation feature.

C

1.0 Float Manager

This section provides reference material for the Palm OS 1.0 float manager. In Palm OS 1.0, the float manager API is declared in the header file `FloatMgr.h`. In Palm OS 2.0, this file was renamed to `FloatMgrOld.h`. In Palm OS 3.5, this file was made private.

NOTE: New applications should no longer use the 1.0 Float Manager. See Chapter 36, "Float Manager," on page 757 for the functions now provided by the Float Manager.

Float Manager Functions

FplAdd Function

Purpose Add two floating-point numbers (returns a + b).

Prototype

```
FloatType FplAdd (FloatType a, FloatType b)
```

Parameters

`a, b`
The floating-point numbers.

Returns Returns the normalized floating-point result of the addition.

Comments Under Palm OS® 2.0 and later, most applications will want to use the arithmetic symbols instead. See the "Floating-Point" section in the *Palm OS Programmer's Companion*, vol. I.

Compatibility This function is not supported by PACE (5.0 New Feature Set or Palm OS Cobalt Feature Set).

FplAToF Function

Purpose Convert a zero-terminated ASCII string to a floating-point number. The string must be in the format: [-]x[.]yyyyyyyy[e[-]zz]

Prototype

```
FloatType FplAToF (char *s)
```

Parameters

`s`
Pointer to the ASCII string.

Returns Returns the floating-point number.

Comments The mantissa of the number is limited to 32 bits.

Compatibility This function is not supported by PACE (5.0 New Feature Set or Palm OS Cobalt Feature Set).

See Also `FplFToA()`, `FplFree()`, `FplInit()`

FplBase10Info Function

Purpose Extract detailed information on the base 10 form of a floating-point number: the base 10 mantissa, exponent, and sign.

Prototype

```
Err FplBase10Info (FloatType a, ULong *mantissaP,
    Int *exponentP, Int *signP)
```

Parameters

`a`
The floating-point number.

`mantissaP`
The base 10 mantissa (return value).

`exponentP`
The base 10 exponent (return value).

`signP`
The sign, 1 or -1 (return value).

Returns Returns an error code, or 0 if no error.

Comments The mantissa is normalized so it contains at least `kMaxSignificantDigits` significant digits when printed as an integer value.

`FlpBase10Info` reports that zero is "negative"; that is, it returns a one for xSign. If this is a problem, a simple workaround is:

```
if (xMantissa == 0)
   xSign = 0;
```

Compatibility This function is not supported by PACE (5.0 New Feature Set or Palm OS Cobalt Feature Set).

FplDiv Function

Purpose Divide two floating-point numbers (result = dividend/divisor).

Prototype

```
FloatType FplDiv (FloatType dividend,
   FloatType divisor)
```

Parameters

`dividend`
Floating-point dividend.

`divisor`
Floating-point divisor.

Returns Returns the normalized floating-point result of the division.

Under Palm OS 2.0 and later, most applications will want to use the arithmetic symbols instead. See the "Floating-Point" section in the *Palm OS Programmer's Companion*, vol. I.

Compatibility This function is not supported by PACE (5.0 New Feature Set or Palm OS Cobalt Feature Set).

FplFloatToLong Function

Purpose Convert a floating-point number to a long integer.

Prototype

```
Long FplFloatToLong (FloatType f)
```

Parameters

`f`
Floating-point number to be converted.

Returns Returns the long integer.

Compatibility This function is not supported by PACE (5.0 New Feature Set or Palm OS Cobalt Feature Set).

See Also FplLongToFloat(), FplFloatToULong()

FplFloatToULong Function

Purpose Convert a floating-point number to an unsigned long integer.

Prototype
```
ULong FplFloatToULong (FloatType f)
```

Parameters

`f`
Floating-point number to be converted.

Returns Returns an unsigned long integer.

Compatibility This function is not supported by PACE (5.0 New Feature Set or Palm OS Cobalt Feature Set).

See Also FplLongToFloat(), FplFloatToLong()

FplFree Function

Purpose Release all memory allocated by the floating-point initialization.

Prototype
```
void FplFree()
```

Parameters `None.`

Returns Returns nothing.

Comments Applications must call this routine after they've called other functions that are part of the float manager.

Compatibility If either 5.0 New Feature Set or Palm OS Cobalt Feature Set is present, this function does nothing.

See Also `FplInit()`

FplFToA Function

Purpose Convert a floating-point number to a zero-terminated ASCII string in exponential format: `[-]x.yyyyyyye[-]zz`

Prototype
```
Err FplFToA (FloatType a, char *s)
```

Parameters

`a`
Floating-point number.

`s`
Pointer to buffer to contain the ASCII string.

Returns Returns an error code, or 0 if no error.

Compatibility This function is not supported by PACE (5.0 New Feature Set or Palm OS Cobalt Feature Set).

See Also `FplAToF()`, `FplFree()`, `FplInit()`

FplInit Function

Purpose Initialize the floating-point conversion routines.

Allocate space in the system heap for floating-point globals.

Initialize the `tenPowers` array in the globals area to the powers of 10 from -99 to +99 in floating-point format.

Prototype

```
Err FplInit()
```

Parameters None.

Returns Returns an error code, or 0 if no error.

Comments Applications must call this routine before calling any other `Fpl` function.

Compatibility If either 5.0 New Feature Set or Palm OS Cobalt Feature Set is present, this function does nothing.

See Also `FplFree()`

FplLongToFloat Function

Purpose Convert a long integer to a floating-point number.

Prototype

```
FloatType FplLongToFloat (Long x)
```

Parameters

`x`
: A long integer.

Returns Returns the floating-point number.

Compatibility This function is not supported by PACE (5.0 New Feature Set or Palm OS Cobalt Feature Set).

FplMul Function

Purpose Multiply two floating-point numbers.

Prototype
```
FloatType FplMul (FloatType a, FloatType b)
```

Parameters
`a, b`
The floating-point numbers.

Returns Returns the normalized floating-point result of the multiplication.

Comments Under Palm OS 2.0 and later, most applications will want to use the arithmetic symbols instead. See the "Floating-Point" section in the *Palm OS Programmer's Companion*, vol. I.

Compatibility This function is not supported by PACE (5.0 New Feature Set or Palm OS Cobalt Feature Set).

FplSub Function

Purpose Subtract two floating-point numbers (returns a - b).

Prototype
```
FloatType FplSub (FloatType a, FloatType b)
```

Parameters
`a, b`
The floating-point numbers.

Returns Returns the normalized floating-point result of the subtraction.

Comments Under Palm OS 2.0 and later, most applications will want to use the arithmetic symbols instead. See the "Floating-Point" section in the *Palm OS Programmer's Companion*, vol. I.

Compatibility This function is not supported by PACE (5.0 New Feature Set or Palm OS Cobalt Feature Set).

D

Palm OS Error Codes

This appendix lists most of the error codes used by the Palm OS, and is sorted by value to serve as an aid to debugging.

Code	Identifier	Description
0x0000	errNone	No error
0x0000	hostErrBase	No error
0x0000	inetErrNone	No error
0x0100	memErrorClass	Memory Manager errors
0x0101	memErrChunkLocked	The associated memory chunk is locked.
0x0102	memErrNotEnoughSpace	A memory error occurred.
0x0103	memErrInvalidParam	One of the function's parameters is invalid or the requested size is too big.
0x0104	memErrChunkNotLocked	
0x0105	memErrCardNotPresent	The specified card can't be found.
0x0106	memErrNoCardHeader	
0x0107	memErrInvalidStoreHeader	The specified card has no storage RAM.
0x0108	memErrRAMOnlyCard	The specified card has no storage RAM.
0x0109	memErrWriteProtect	
0x010a	memErrNoRAMOnCard	
0x010b	memErrNoStore	
0x010c	memErrROMOnlyCard	
0x0200	dmErrorClass	Data Manager errors

Code	Identifier	Description
0x0201	dmErrMemError	A memory error occurred.
0x0202	dmErrIndexOutOfRange	The specified index is out of range.
0x0203	dmErrInvalidParam	The function received an invalid parameter.
0x0204	dmErrReadOnly	You've attempted to write to or modify a database that is in read-only mode.
0x0205	dmErrDatabaseOpen	The function cannot be performed on an open database, and the database is open.
0x0206	dmErrCantOpen	The database cannot be opened.
0x0207	dmErrCantFind	The specified resource can't be found.
0x0208	dmErrRecordInWrongCard	You've attempted to attach a record to a database when the record and database reside on different memory cards.
0x0209	dmErrCorruptDatabase	The database is corrupted.
0x020a	dmErrRecordDeleted	The record has been deleted.
0x020b	dmErrRecordArchived	The function requires that the record not be archived, but it is.
0x020c	dmErrNotRecordDB	You've attempted to perform a record function on a resource database.
0x020d	dmErrNotResourceDB	You've attempted to perform a resource manager function on a record database.
0x020e	dmErrROMBased	You've attempted to delete or modify a ROM-based database.
0x020f	dmErrRecordBusy	The function requires that the record not be busy, but it is.
0x0210	dmErrResourceNotFound	The resource can't be found.
0x0211	dmErrNoOpenDatabase	The function is to search all open databases, but there are none.

Code	Identifier	Description
0x0212	dmErrInvalidCategory	
0x0213	dmErrNotValidRecord	The record handle is invalid.
0x0214	dmErrWriteOutOfBounds	A write operation exceeded the bounds of the record.
0x0215	dmErrSeekFailed	The operation of seeking the next record in the category failed.
0x0216	dmErrAlreadyOpenForWrites	The database is already open with write access.
0x0217	dmErrOpenedByAnotherTask	You've attempted to open a database that another task already has open.
0x0218	dmErrUniqueIDNotFound	A record with the specified unique ID can't be found.
0x0219	dmErrAlreadyExists	Another database with the same name already exists in RAM store.
0x021a	dmErrInvalidDatabaseName	The name you've specified for the database is invalid.
0x021b	dmErrDatabaseProtected	The database is marked as protected.
0x021c	dmErrDatabaseNotProtected	`DmDatabaseProtect()` failed to protect the specified database.
0x0300	serErrorClass	Serial Manager errors
0x0301	serErrBadParam	One of the function's parameters is invalid. See the Serial Manager function's description for additional detail.
0x0302	serErrBadPort	The specified port doesn't exist.
0x0303	serErrNoMem	
0x0304	serErrBadConnID	
0x0305	serErrTimeOut	Unable to receive data within the specified timeout period.

Code	Identifier	Description
0x0306	serErrLineErr	A line error occurred during the receipt of data. Use `SrmGetStatus()` to obtain the exact line error.
0x0307	serErrAlreadyOpen	The Serial Manager already has a port open, or the specified port already has an installed foreground or background owner.
0x0308	serErrStillOpen	
0x0309	serErrNotOpen	The serial port is not open.
0x030a	serErrNotSupported	The specified port is not the foreground port, the specified op code is not supported in the current configuration, the port is a yielded port, or the feature isn't supported by the hardware.
0x030b	serErrNoDevicesAvail	No serial devices could be found.
0x030c	serErrConfigurationFailed	The port needs time to configure, and the configuration has failed.
0x0400	slkErrorClass	Serial Link Manager errors
0x0401	slkErrChecksum	
0x0402	slkErrFormat	
0x0403	slkErrBuffer	
0x0404	slkErrTimeOut	
0x0405	slkErrHandle	
0x0406	slkErrBodyLimit	
0x0407	slkErrTransId	
0x0408	slkErrResponse	
0x0409	slkErrNoDefaultProc	
0x040a	slkErrWrongPacketType	

Code	Identifier	Description
0x040b	slkErrBadParam	
0x040c	slkErrAlreadyOpen	
0x040d	slkErrOutOfSockets	
0x040e	slkErrSocketNotOpen	
0x040f	slkErrWrongDestSocket	
0x0410	slkErrWrongPktType	
0x0411	slkErrBusy	
0x0412	slkErrNotOpen	
0x0500	sysErrorClass	System Manager errors
0x0501	sysErrTimeout	
0x0502	sysErrParamErr	
0x0503	sysErrNoFreeResource	
0x0504	sysErrNoFreeRAM	
0x0505	sysErrNotAllowed	
0x0506	sysErrSemInUse	
0x0507	sysErrInvalidID	
0x0508	sysErrOutOfOwnerIDs	
0x0509	sysErrNoFreeLibSlots	
0x050a	sysErrLibNotFound	The Exchange Manager couldn't find library or application to respond to URL.
0x050b	sysErrDelayWakened	
0x050c	sysErrRomIncompatible	
0x050d	sysErrBufTooSmall	
0x050e	sysErrPrefNotFound	
0x0510	sysNotifyErrEntryNotFound	

Code	Identifier	Description
0x0511	sysNotifyErrDuplicateEntry	
0x0513	sysNotifyErrBroadcastBusy	
0x0514	sysNotifyErrBroadcastCancelled	
0x0515	sysErrMbId	
0x0516	sysErrMbNone	
0x0517	sysErrMbBusy	
0x0518	sysErrMbFull	
0x0519	sysErrMbDepth	
0x051a	sysErrMbEnv	
0x051b	sysNotifyErrQueueFull	
0x051c	sysNotifyErrQueueEmpty	
0x051d	sysNotifyErrNoStackSpace	
0x051e	sysErrNotInitialized	
0x051f	sysErrNotAsleep	
0x0520	sysErrNotAsleepN	
0x0600	fplErrorClass	Old Floating Point Library errors
0x0680	flpErrorClass	Floating Point Library errors
0x0681	flpErrOutOfRange	
0x0700	evtErrorClass	System Event Manager errors
0x0701	evtErrParamErr	
0x0702	evtErrQueueFull	
0x0703	evtErrQueueEmpty	
0x0800	sndErrorClass	Sound Manager errors
0x0801	sndErrBadParam	

Code	Identifier	Description
0x0802	sndErrBadChannel	
0x0803	sndErrMemory	
0x0804	sndErrOpen	
0x0805	sndErrQFull	
0x0806	sndErrQEmpty	
0x0807	sndErrFormat	
0x0808	sndErrBadStream	
0x0809	sndErrInterrupted	
0x080a	sndErrNotImpl	
0x080b	sndErrInvalidStream	
0x0900	almErrorClass	Alarm Manager errors
0x0901	almErrMemory	There is insufficient memory to complete the requested operation.
0x0902	almErrFull	The alarm table is full.
0x0a00	timErrorClass	Time Manager errors
0x0a01	timErrMemory	
0x0b00	penErrorClass	Pen Manager errors
0x0b01	penErrBadParam	
0x0b02	penErrIgnorePoint	
0x0c00	ftrErrorClass	Feature Manager errors
0x0c01	ftrErrInvalidParam	
0x0c02	ftrErrNoSuchFeature	The specified feature number doesn't exist for the specified creator, the index is out of range, or some other error exists. See the Feature Manager function's description for more details.

Code	Identifier	Description
0x0c03	ftrErrAlreadyExists	
0x0c04	ftrErrROMBased	
0x0c05	ftrErrInternalErr	
0x0d00	cmpErrorClass	Connection Manager (HotSync) errors
0x0e00	dlkErrorClass	Desktop Link Manager errors
0x0e01	dlkErrParam	
0x0e02	dlkErrMemory	
0x0e03	dlkErrNoSession	
0x0e04	dlkErrSizeErr	
0x0e05	dlkErrLostConnection	
0x0e06	dlkErrInterrupted	
0x0e07	dlkErrUserCan	
0x0e08	dlkErrIncompatibleProducts	
0x0e09	dlkErrNPOD	
0x0f00	padErrorClass	PAD Manager errors
0x1000	grfErrorClass	Graffiti Manager errors
0x1001	grfErrBadParam	
0x1002	grfErrPointBufferFull	
0x1003	grfErrNoGlyphTable	
0x1004	grfErrNoDictionary	
0x1005	grfErrNoMapping	
0x1006	grfErrMacroNotFound	
0x1007	grfErrDepthTooDeep	
0x1008	grfErrMacroPtrTooSmall	

Code	Identifier	Description
0x1009	grfErrNoMacros	
0x1010	grfErrGenericHWRErrBase	
0x1081	grfErrMacroIncomplete	
0x1082	grfErrBranchNotFound	
0x1100	mdmErrorClass	Modem Manager errors
0x1101	mdmErrNoTone	
0x1102	mdmErrNoDCD	
0x1103	mdmErrBusy	
0x1104	mdmErrUserCan	
0x1105	mdmErrCmdError	
0x1106	mdmErrNoModem	
0x1107	mdmErrMemory	
0x1108	mdmErrPrefs	
0x1109	mdmErrDial	
0x110a	mdmErrNoPhoneNum	
0x1200	netErrorClass	Net Library errors
0x1201	netErrAlreadyOpen	
0x1202	netErrNotOpen	
0x1203	netErrStillOpen	
0x1204	netErrParamErr	
0x1205	netErrNoMoreSockets	
0x1206	netErrOutOfResources	
0x1207	netErrOutOfMemory	
0x1208	netErrSocketNotOpen	

Code	Identifier	Description
0x1209	netErrSocketBusy	
0x120a	netErrMessageTooBig	
0x120b	netErrSocketNotConnected	
0x120c	netErrNoInterfaces	
0x120d	netErrBufTooSmall	
0x120e	netErrUnimplemented	
0x120f	netErrPortInUse	
0x1210	netErrQuietTimeNotElapsed	
0x1211	netErrInternal	
0x1212	netErrTimeout	
0x1213	netErrSocketAlreadyConnected	
0x1214	netErrSocketClosedByRemote	
0x1215	netErrOutOfCmdBlocks	
0x1216	netErrWrongSocketType	
0x1217	netErrSocketNotListening	
0x1218	netErrUnknownSetting	
0x1219	netErrInvalidSettingSize	
0x121a	netErrPrefNotFound	
0x121b	netErrInvalidInterface	
0x121c	netErrInterfaceNotFound	
0x121d	netErrTooManyInterfaces	
0x121e	netErrBufWrongSize	
0x121f	netErrUserCancel	
0x1220	netErrBadScript	

Code	Identifier	Description
0x1221	netErrNoSocket	
0x1222	netErrSocketRcvBufFull	
0x1223	netErrNoPendingConnect	
0x1224	netErrUnexpectedCmd	
0x1225	netErrNoTCB	
0x1226	netErrNilRemoteWindowSize	
0x1227	netErrNoTimerProc	
0x1228	netErrSocketInputShutdown	
0x1229	netErrCmdBlockNotCheckedOut	
0x122a	netErrCmdNotDone	
0x122b	netErrUnknownProtocol	
0x122c	netErrUnknownService	
0x122d	netErrUnreachableDest	
0x122e	netErrReadOnlySetting	
0x122f	netErrWouldBlock	
0x1230	netErrAlreadyInProgress	
0x1231	netErrPPPTimeout	
0x1232	netErrPPPBroughtDown	
0x1233	netErrAuthFailure	
0x1234	netErrPPPAddressRefused	
0x1235	netErrDNSNameTooLong	
0x1236	netErrDNSBadName	
0x1237	netErrDNSBadArgs	
0x1238	netErrDNSLabelTooLong	

Code	Identifier	Description
0x1239	netErrDNSAllocationFailure	
0x123a	netErrDNSTimeout	
0x123b	netErrDNSUnreachable	
0x123c	netErrDNSFormat	
0x123d	netErrDNSServerFailure	
0x123e	netErrDNSNonexistantName	
0x123f	netErrDNSNIY	
0x1240	netErrDNSRefused	
0x1241	netErrDNSImpossible	
0x1242	netErrDNSNoRRS	
0x1243	netErrDNSAborted	
0x1244	netErrDNSBadProtocol	
0x1245	netErrDNSTruncated	
0x1246	netErrDNSNoRecursion	
0x1247	netErrDNSIrrelevant	
0x1248	netErrDNSNotInLocalCache	
0x1249	netErrDNSNoPort	
0x124a	netErrIPCantFragment	
0x124b	netErrIPNoRoute	
0x124c	netErrIPNoSrc	
0x124d	netErrIPNoDst	
0x124e	netErrIPktOverflow	
0x124f	netErrTooManyTCPConnections	
0x1250	netErrNoDNSServers	

Code	Identifier	Description
0x1251	netErrInterfaceDown	
0x1252	netErrNoChannel	
0x1253	netErrDieState	
0x1254	netErrReturnedInMail	
0x1255	netErrReturnedNoTransfer	
0x1256	netErrReturnedIllegal	
0x1257	netErrReturnedCongest	
0x1258	netErrReturnedError	
0x1259	netErrReturnedBusy	
0x125a	netErrGMANState	
0x125b	netErrQuitOnTxFail	
0x125c	netErrFlexListFull	
0x125d	netErrSenderMAN	
0x125e	netErrIllegalType	
0x125f	netErrIllegalState	
0x1260	netErrIllegalFlags	
0x1261	netErrIllegalSendlist	
0x1262	netErrIllegalMPAKLength	
0x1263	netErrIllegalAddressee	
0x1264	netErrIllegalPacketClass	
0x1265	netErrBufferLength	
0x1266	netErrNiCdLowBattery	
0x1267	netErrRFinterfaceFatal	
0x1268	netErrIllegalLogout	

Code	Identifier	Description
0x1269	netErrAAARadioLoad	
0x126a	netErrAntennaDown	
0x126b	netErrNiCdCharging	
0x126c	netErrAntennaWentDown	
0x126d	netErrNotActivated	
0x126e	netErrRadioTemp	
0x126f	netErrNiCdChargeError	
0x1270	netErrNiCdSag	
0x1271	netErrNiCdChargeSuspend	
0x1273	netErrConfigNotFound	
0x1274	netErrConfigCantDelete	
0x1275	netErrConfigTooMany	
0x1276	netErrConfigBadName	
0x1277	netErrConfigNotAlias	
0x1278	netErrConfigCantPointToAlias	
0x1279	netErrConfigEmpty	
0x127a	netErrAlreadyOpenWithOther Config	
0x127b	netErrConfigAliasErr	
0x127c	netErrNoMultiPktAddr	
0x127d	netErrOutOfPackets	
0x127e	netErrMultiPktAddrReset	
0x127f	netErrStaleMultiPktAddr	
0x1280	netErrScptPluginMissing	
0x1281	netErrScptPluginLaunchFail	

Code	Identifier	Description
0x1282	netErrScptPluginCmdFail	
0x1283	netErrScptPluginInvalidCmd	
0x1284	netErrTelMissingComponent	
0x1285	netErrTelErrorNotHandled	
0x1286	netErrDeviceNotReady	
0x1287	netErrDeviceInitFail	
0x1288	netErrInterfaceIncompatible	
0x1300	htalErrorClass	HTAL Library errors
0x1400	inetErrorClass	INet Library errors
0x1401	inetErrTooManyClients	
0x1402	inetErrHandleInvalid	
0x1403	inetErrParamsInvalid	
0x1404	inetErrURLVersionInvalid	
0x1405	inetErrURLBufTooSmall	
0x1406	inetErrURLInvalid	
0x1407	inetErrTooManySockets	
0x1408	inetErrNoRequestCreated	
0x1409	inetErrNotConnected	
0x140a	inetErrInvalidRequest	
0x140b	inetErrNeedTime	
0x140c	inetErrHostnameInvalid	
0x140d	inetErrInvalidPort	
0x140e	inetErrInvalidHostAddr	
0x140f	inetErrNilBuffer	

Code	Identifier	Description
0x1410	inetErrConnectTimeout	
0x1411	inetErrResolveTimeout	
0x1412	inetErrSendReqTimeout	
0x1413	inetErrReadTimeout	
0x1414	inetErrBufTooSmall	
0x1415	inetErrSchemeNotSupported	
0x1416	inetErrInvalidResponse	
0x1419	inetErrSettingTooLarge	
0x141a	inetErrSettingSizeInvalid	
0x141b	inetErrRequestTooLong	
0x141c	inetErrSettingNotImplemented	
0x141d	inetErrConfigNotFound	
0x141e	inetErrConfigCantDelete	
0x141f	inetErrConfigTooMany	
0x1420	inetErrConfigBadName	
0x1421	inetErrConfigNotAlias	
0x1422	inetErrConfigCantPointToAlias	
0x1423	inetErrConfigEmpty	
0x1425	inetErrConfigAliasErr	
0x1426	inetErrNoWirelessInterface	
0x1427	inetErrEncryptionNotAvail	
0x1428	inetErrNeedRetryEncSeqNum	
0x1429	inetErrNeedRetryEncPublicKey	
0x142a	inetErrResponseTooShort	

Code	Identifier	Description
0x142b	inetErrMobitexIllegalOKHost	
0x142c	inetErrMobitexIllegalBadHost	
0x142d	inetErrHTTPBadRequest	
0x142e	inetErrHTTPUnauthorized	
0x142f	inetErrHTTPForbidden	
0x1430	inetErrHTTPNotFound	
0x1431	inetErrHTTPMethodNotAllowed	
0x1432	inetErrHTTPNotAcceptable	
0x1433	inetErrHTTPProxyAuthRequired	
0x1434	inetErrHTTPRequestTimeout	
0x1435	inetErrHTTPConflict	
0x1436	inetErrHTTPGone	
0x1437	inetErrHTTPLengthRequired	
0x1438	inetErrHTTPPreconditionFailed	
0x1439	inetErrHTTPRequestTooLarge	
0x143a	inetErrHTTPRequestURITooLong	
0x143b	inetErrHTTPUnsupportedType	
0x143c	inetErrHTTPServerError	
0x143d	inetErrCTPServerError	
0x143e	inetErrTypeNotCached	
0x143f	inetErrCacheInvalid	
0x1440	inetErrURLDispatched	

Code	Identifier	Description
0x1441	inetErrDatabaseNotFound	
0x1442	inetErrCTPMalformedRequest	
0x1443	inetErrCTPUnknownCommand	
0x1444	inetErrCTPTruncated	
0x1445	inetErrCTPUnknownError	
0x1446	inetErrCTPProxyError	
0x1447	inetErrCTPSocketErr	
0x1448	inetErrCTPInvalidURL	
0x1449	inetErrCTPReferringPageOutOfDate	
0x144a	inetErrCTPBadRequest	
0x144b	inetErrUNUSED	
0x144c	inetErrCTPMailServerDown	
0x144d	inetErrCTPHostNotFound	
0x144e	inetErrCTPContentInvalidTag	
0x144f	inetErrCTPContentInternal	
0x1450	inetErrCTPContentDataEnd	
0x1451	inetErrCTPContentResourceTooBig	
0x1452	inetErrCTPContentNoNoFrames	
0x1453	inetErrCTPContentUnsupportedContent	
0x1454	inetErrCTPContentUnsupportedEncoding	

Code	Identifier	Description
0x1455	inetErrCTPContentBadForm	
0x1456	inetErrCTPContentBadFormMissingAction	
0x1457	inetErrCTPContentBadFormMissingMethod	
0x1458	inetErrCTPContentNoSourceData	
0x1459	inetErrCTPContentBadImage	
0x145a	inetErrCTPContentImageTooLarge	
0x145b	inetErrMobitexErrorHandled	
0x145c	inetErrProxyDownBadHost	
0x145d	inetErrHostConnectionLost	
0x145e	inetErrLinkNotFound	
0x145f	inetErrCacheInvalidSort	
0x1500	exgErrorClass	Exchange Manager errors
0x1501	exgMemError	There isn't enough free memory to respond to the request; or the Exchange Manager ouldn't read the data to be sent.
0x1502	exgErrStackInit	
0x1503	exgErrUserCancel	The user cancelled the connection operation.
0x1504	exgErrNoReceiver	
0x1505	exgErrNoKnownTarget	There is no default application or library for the specified type of data.
0x1506	exgErrTargetMissing	The `target` field contains a creator ID, but the application with that creator ID does not exist.

Code	Identifier	Description
0x1507	exgErrNotAllowed	
0x1508	exgErrBadData	
0x1509	exgErrAppError	
0x150a	exgErrUnknown	
0x150b	exgErrDeviceFull	
0x150c	exgErrDisconnected	
0x150d	exgErrNotFound	
0x150e	exgErrBadParam	
0x150f	exgErrNotSupported	A preview is in progress, and the exchange library identified by `libraryRef` doesn't support preview mode; or the library doesn't support the operation specified in `socketP`; or the exchange library does not support the requested operation. See the Exchange Manager function's description for more detail.
0x1510	exgErrDeviceBusy	
0x1511	exgErrBadLibrary	The default exchange library couldn't be found.
0x1512	exgErrNotEnoughPower	The battery does not have enough power to perform the operation.
0x1600	fileErrorClass	File Stream Manager errors
0x1601	fileErrMemError	
0x1602	fileErrInvalidParam	
0x1603	fileErrCorruptFile	
0x1604	fileErrNotFound	
0x1605	fileErrTypeCreatorMismatch	

Code	Identifier	Description
0x1606	fileErrReplaceError	
0x1607	fileErrCreateError	
0x1608	fileErrOpenError	
0x1609	fileErrInUse	
0x160a	fileErrReadOnly	
0x160b	fileErrInvalidDescriptor	
0x160c	fileErrCloseError	
0x160d	fileErrOutOfBounds	
0x160e	fileErrPermissionDenied	
0x160f	fileErrIOError	
0x1610	fileErrEOF	
0x1611	fileErrNotStream	
0x1700	rfutErrorClass	RFUT Library errors
0x1800	txtErrorClass	Text Manager errors
0x1801	txtErrUknownTranslitOp	
0x1802	txtErrTranslitOverrun	
0x1803	txtErrTranslitOverflow	
0x1804	txtErrConvertOverflow	
0x1805	txtErrConvertUnderflow	
0x1806	txtErrUnknownEncoding	
0x1807	txtErrNoCharMapping	
0x1808	txtErrTranslitUnderflow	
0x1809	txtErrMalformedText	
0x180a	txtErrUnknownEncodingFallbackCopy	

Code	Identifier	Description
0x1900	tsmErrorClass	Text Services Manager errors
0x1a00	webErrorClass	Web Library errors
0x1b00	secErrorClass	Security Library errors
0x1c00	emuErrorClass	Emulator Control Manager errors
0x1c00	hostErrorClass	Host Control Manager errors
0x1c01	hostErrUnknownGestaltSelector	
0x1c02	hostErrDiskError	
0x1c03	hostErrOutOfMemory	
0x1c04	hostErrMemReadOutOfRange	
0x1c05	hostErrMemWriteOutOfRange	
0x1c06	hostErrMemInvalidPtr	
0x1c07	hostErrInvalidParameter	
0x1c08	hostErrTimeout	
0x1c09	hostErrInvalidDeviceType	
0x1c0a	hostErrInvalidRAMSize	
0x1c0b	hostErrFileNotFound	
0x1c0c	hostErrRPCCall	
0x1c0d	hostErrSessionRunning	
0x1c0e	hostErrSessionNotRunning	
0x1c0f	hostErrNoSignalWaiters	
0x1c10	hostErrSessionNotPaused	
0x1c11	hostErrPermissions	
0x1c12	hostErrFileNameTooLong	
0x1c13	hostErrNotADirectory	

Code	Identifier	Description
0x1c14	hostErrTooManyFiles	
0x1c15	hostErrFileTooBig	
0x1c16	hostErrReadOnlyFS	
0x1c17	hostErrIsDirectory	
0x1c18	hostErrExists	
0x1c19	hostErrOpNotAvailable	
0x1c1a	hostErrDirNotEmpty	
0x1c1b	hostErrDiskFull	
0x1c1c	hostErrUnknownError	
0x1c1d	hostErrProfilingNotReady	
0x1d00	flshErrorClass	Flash Manager errors
0x1e00	pwrErrNone	
0x1e00	pwrErrorClass	Power Manager errors
0x1e01	pwrErrBacklight	
0x1e02	pwrErrRadio	
0x1e03	pwrErrBeam	
0x1e04	pwrErrGeneric	
0x1f00	cncErrorClass	Connection Manager (serial communication) errors
0x1f01	cncErrAddProfileFailed	
0x1f01	kCncErrAddProfileFailed	
0x1f02	cncErrProfileListFull	
0x1f02	kCncErrProfileListFull	
0x1f03	cncErrGetProfileFailed	
0x1f03	kCncErrGetProfileFailed	

Code	Identifier	Description
0x1f04	cncErrConDBNotFound	
0x1f04	kCncErrDBAccessFailed	
0x1f05	cncErrGetProfileListFailed	
0x1f05	kCncErrGetProfileListFailed	
0x1f06	cncErrProfileReadOnly	
0x1f06	kCncErrProfileReadOnly	
0x1f07	cncErrProfileNotFound	
0x1f07	kCncErrProfileNotFound	
0x1f08	kCncErrProfileParamNotFound	
0x1f09	kCncErrProfileParamReadOnly	
0x1f0a	kCncErrProfileParamNameHas Change	
0x1f0b	kCncErrProfileGetParamFailed	
0x1f0c	kCncErrProfileSetParamFailed	
0x1f0d	kCncErrProfileBadParamSize	
0x1f0e	kCncErrProfileBadSystemFlagB itnum	
0x2000	actvErrorClass	Activation application errors
0x2100	radioErrorClass	Radio Manager (library) errors
0x2200	dispErrorClass	Display Driver errors
0x2300	bltErrorClass	Blitter Driver errors
0x2400	winErrorClass	Window Manager errors
0x2401	winErrPalette	
0x2500	omErrorClass	Overlay Manager errors

Code	Identifier	Description
0x2501	omErrBaseRequiresOverlay	An attempt was made to open a stripped resource database, but no associated overlay could be found.
0x2502	omErrUnknownLocale	An attempt was made to open a resource database with overlays using an unknown locale.
0x2503	omErrBadOverlayDBName	
0x2504	omErrInvalidLocaleIndex	
0x2505	omErrInvalidLocale	
0x2506	omErrInvalidSystemOverlay	
0x2507	omErrNoNextSystemLocale	
0x2600	menuErrorClass	Menu Manager errors
0x2601	menuErrNoMenu	The `textP` parameter is `NULL`.
0x2602	menuErrNotFound	The menu doesn't contain an item with the specified ID.
0x2603	menuErrSameId	The menu already contains an item with the specified ID
0x2604	menuErrTooManyItems	The command toolbar already has the maximum number of buttons allowed.
0x2605	menuErrOutOfMemory	There is not enough memory available to perform the operation.
0x2700	lz77ErrorClass	LZ77 Library errors
0x2700	lz77ErrNonFatalFirstErr	
0x2701	lz77ErrNonFatalInputBufferIncomplete	
0x2702	lz77ErrNonFatalOutputBufferFull	
0x277f	lz77ErrNonFatalLastErr	

Code	Identifier	Description
0x2780	lz77ErrFatalFirstErr	
0x2780	lz77ErrFatalUnfinishedInputBuffer	
0x2781	lz77ErrFatalInputBufferIncomplete	
0x2782	lz77ErrFatalInputBufferInvalid	
0x2783	lz77ErrFatalMemAllocation	
0x2784	lz77ErrFatalHandleInvalid	
0x2785	lz77ErrFatalCantChangeToCompress	
0x2786	lz77ErrFatalUnknownVersion	
0x2787	lz77ErrFatalOutputBufferTooSmall	
0x2788	lz77ErrFatalInvalidArgument	
0x27ff	lz77ErrFatalLastErr	
0x2800	smsErrorClass	SMS Library errors
0x2801	kSmsErrMaxSizeExceeded	
0x2900	expErrorClass	Expansion Manager and Slot Driver Library errors
0x2901	expErrUnsupportedOperation	The operation is unsupported or undefined.
0x2902	expErrNotEnoughPower	The required power is not available.
0x2903	expErrCardNotPresent	There is no card present in the given slot.
0x2904	expErrInvalidSlotRefNum	The slot reference number is not valid.
0x2905	expErrSlotDeallocated	The slot reference number is within the valid range, but the slot has been deallocated.

Code	Identifier	Description
0x2906	expErrCardNoSectorReadWrite	The card does not support the slot driver block read/write API.
0x2907	expErrCardReadOnly	The card supports the slot driver block read/write API but the card is read only.
0x2908	expErrCardBadSector	The card supports the slot driver block read/write API but the sector is bad.
0x2909	expErrCardProtectedSector	The card supports the slot driver block read/write API but the sector is protected.
0x290a	expErrNotOpen	The slot driver library has not been opened.
0x290b	expErrStillOpen	The slot driver library is still open; it may have been opened more than once.
0x290c	expErrUnimplemented	The call is unimplemented.
0x290d	expErrEnumerationEmpty	There are no values remaining to enumerate.
0x290e	expErrIncompatibleAPIVer	The API version of the underlying slot driver is not supported by this version of Expansion Manager.
0x2a00	vfsErrorClass	Virtual Filesystem Manager and Filesystem Library errors
0x2a01	vfsErrBufferOverflow	The supplied buffer is too small.
0x2a02	vfsErrFileGeneric	Generic file error.
0x2a03	vfsErrFileBadRef	The file reference is invalid: it has been closed or was not obtained from `VFSFileOpen()`.
0x2a04	vfsErrFileStillOpen	Returned from the underlying file system's delete function if the file is still open.

Code	Identifier	Description
0x2a05	vfsErrFilePermissionDenied	The requested permissions could not be granted.
0x2a06	vfsErrFileAlreadyExists	A file with this name exists already in this location.
0x2a07	vfsErrFileEOF	The file pointer is at the end of the file.
0x2a08	vfsErrFileNotFound	The file was not found at the specified location.
0x2a09	vfsErrVolumeBadRef	The volume reference number is invalid.
0x2a0a	vfsErrVolumeStillMounted	Returned from the underlying file system's format function if the volume is still mounted.
0x2a0b	vfsErrNoFileSystem	None of the installed file systems support this operation.
0x2a0c	vfsErrBadData	The operation could not be completed because of invalid data.
0x2a0d	vfsErrDirNotEmpty	The directory is not empty and therefore cannot be deleted.
0x2a0e	vfsErrBadName	Invalid filename, path, or volume label.
0x2a0f	vfsErrVolumeFull	There is insufficient space left on the volume.
0x2a10	vfsErrUnimplemented	
0x2a11	vfsErrNotADirectory	This operation can only performed on a directory.
0x2a12	vfsErrIsADirectory	This operation can only be performed on a regular file, not a directory.
0x2a13	vfsErrDirectoryNotFound	Returned when the path leading up to the file does not exist.

Code	Identifier	Description
0x2a14	vfsErrNameShortened	A volume name or filename was automatically shortened to conform to the file system specification.
0x2b00	lmErrorClass	Locale Manager errors
0x2b01	lmErrUnknownLocale	The locale could not be found.
0x2b02	lmErrBadLocaleIndex	The locale index is out of range.
0x2b03	lmErrBadLocaleSettingChoice	The locale setting choice parameter contains an unknown or unsupported value.
0x2b04	lmErrSettingDataOverflow	The output buffer is too small to hold the setting's value.
0x2c00	intlErrorClass	International Manager errors
0x2c01	intlErrInvalidSelector	
0x2d00	pdiErrorClass	PDI Library errors
0x2d01	pdiErrRead	
0x2d02	pdiErrWrite	
0x2d03	pdiErrNoPropertyName	
0x2d04	pdiErrNoPropertyValue	
0x2d05	pdiErrMoreChars	
0x2d06	pdiErrNoMoreFields	
0x2d07	pdiErrOpenFailed	
0x2d08	pdiErrCloseFailed	
0x2e00	attnErrorClass	Attention Manager errors
0x2e01	attnErrMemory	There is insufficient memory to perform the requested operation.
0x2f00	telErrorClass	Telephony Manager errors

Code	Identifier	Description
0x2f01	telErrMsgAllocation	Couldn't allocate message.
0x2f02	telErrUnknown	Unknown Telephony internal error.
0x2f03	telErrMemAllocation	Memory allocation error.
0x2f04	telErrResultTimeOut	Time-out was reached.
0x2f05	telErrResultUserCancel	User cancelled action.
0x2f06	telErrResultBusyResource	Resource is busy.
0x2f07	telErrInvalidAppId	Don’t know that application.
0x2f08	telErrTooManyApps	Applications table is full.
0x2f09	telErrSecurity	Access to ME has not been granted.
0x2f0a	telErrBufferSize	Buffer used to retrieve data is too small.
0x2f0b	telErrFeatureNotSupported	The feature is not supported by the phone or network.
0x2f0c	telErrPhoneComm	The communication link with the phone is down.
0x2f0d	telErrPhoneReply	The phone reply syntax is incorrect. Check the phone driver.
0x2f0e	telErrCommandFailed	The phone couldn’t achieve the associated command. Check the phone driver.
0x2f0f	telErrSpcLineIsBusy	Spc call failure events, error field values.
0x2f10	telErrPhoneCodeRequired	Phone code required.
0x2f11	telErrNoSIMInserted	No SIM inserted.
0x2f12	telErrPINRequired	PIN is required.
0x2f13	telErrPUKRequired	PUK is required.
0x2f14	telErrSIMFailure	The SIM is not working properly.
0x2f15	telErrSIMBusy	The SIM couldn’t reply.

Code	Identifier	Description
0x2f16	telErrSIMWrong	The SIM is not accepted by the phone.
0x2f17	telErrPassword	Incorrect password.
0x2f18	telErrPIN2Required	PIN2 is required.
0x2f19	telErrPUK2Required	PUK2 is required.
0x2f1a	telErrPhoneMemAllocation	Phone memory is full.
0x2f1b	telErrInvalidIndex	Invalid index when accessing a storage.
0x2f1c	telErrEntryNotFound	Entry not found.
0x2f1d	telErrPhoneMemFailure	The phone encountered a memory error.
0x2f1e	telErrInvalidString	Bad character in text string.
0x2f1f	telErrInvalidDial	Bad character in dial string.
0x2f20	telErrNoNetwork	No network available.
0x2f21	telErrNetworkTimeOut	The network didn't reply within the "normal" time delay.
0x2f22	telErrInvalidParameter	Bad parameter passed to an API.
0x2f23	telErrValidityPeriod	The specified short message validity period is invalid.
0x2f24	telErrCodingScheme	The specified short message coding scheme is invalid.
0x2f25	telErrPhoneNumber	The specified short message smsc or destination phone number is invalid.
0x2f26	telErrValueStale	Information couldn't be retrieved. A copy of the last retrieved value was returned.
0x2f27	telErrTTaskNotRunning	The Telephony task is not running.
0x2f28	telErrPhoneToSIMPINRequired	Phone 2 SIM PIN is required.
0x2f29	telErrSpecificDrvNotFound	The specified driver was not found.
0x2f2a	telErrGenericDrvNotFound	The generic driver was not found.

Code	Identifier	Description
0x2f2b	telErrNoSpecificDrv	No specific driver was specified.
0x2f2c	telErrSpcLineIsReleased	The call has been released.
0x2f2d	telErrSpcCallError	The call has encountered an error.
0x2f2e	telErrNotInstalled	The shared library couldn't be installed.
0x2f2f	telErrVersion	The shared library version doesn't match the application's.
0x2f30	telErrSettings	Bad telephony settings: Phone panel preferences don't exist of the telephony profile is not correctly set.
0x2f31	telErrUnavailableValue	The requested value can't be retrieved at that time (for instance, TelSpcGetCallerNumber and no active line).
0x2f32	telErrLimitedCompatibility	The current driver is partially compatible with the connected phone.
0x2f33	telErrProfileConflict	The currently used profile conflicts with the requested profile.
0x2f34	telErrLibStillInUse	The shared library is currently being used by another application. Don't unload it.
0x2f35	telErrTTaskNotFound	Couldn't find the specified (by phone driver) telephony task.
0x3000	hwrErrorClass	Hardware Manager (HAL) errors
0x3100	blthErrorClass	Bluetooth Library errors
0x3101	btLibErrNotOpen	
0x3102	btLibErrBluetoothOff	
0x3103	btLibErrNoPrefs	
0x3104	btLibErrAlreadyOpen	
0x3105	btLibErrOutOfMemory	

Code	Identifier	Description
0x3106	btLibErrFailed	
0x3107	btLibErrInProgress	
0x3108	btLibErrParamError	
0x3109	btLibErrTooMany	
0x310a	btLibErrPending	
0x310b	btLibErrNotInProgress	
0x310c	btLibErrRadioInitFailed	
0x310d	btLibErrRadioFatal	
0x310e	btLibErrRadioInitialized	
0x310f	btLibErrRadioSleepWake	
0x3110	btLibErrNoConnection	
0x3111	btLibErrAlreadyRegistered	
0x3112	btLibErrNoAclLink	
0x3113	btLibErrSdpRemoteRecord	
0x3114	btLibErrSdpAdvertised	
0x3115	btLibErrSdpFormat	
0x3116	btLibErrSdpNotAdvertised	
0x3117	btLibErrSdpQueryVersion	
0x3118	btLibErrSdpQueryHandle	
0x3119	btLibErrSdpQuerySyntax	
0x311a	btLibErrSdpQueryPduSize	
0x311b	btLibErrSdpQueryContinuation	
0x311c	btLibErrSdpQueryResources	
0x311d	btLibErrSdpQueryDisconnect	

Code	Identifier	Description
0x311e	btLibErrSdpInvalidResponse	
0x311f	btLibErrSdpAttributeNotSet	
0x3120	btLibErrSdpMapped	
0x3121	btLibErrSocket	
0x3122	btLibErrSocketProtocol	
0x3123	btLibErrSocketRole	
0x3124	btLibErrSocketPsmUnavailable	
0x3125	btLibErrSocketChannelUnavailable	
0x3126	btLibErrSocketUserDisconnect	
0x3127	btLibErrCanceled	
0x3128	btLibErrBusy	
0x3129	btLibMeStatusUnknownHciCommand	
0x312a	btLibMeStatusNoConnection	
0x312b	btLibMeStatusHardwareFailure	
0x312c	btLibMeStatusPageTimeout	
0x312d	btLibMeStatusAuthenticateFailure	
0x312e	btLibMeStatusMissingKey	
0x312f	btLibMeStatusMemoryFull	
0x3130	btLibMeStatusConnnectionTimeout	
0x3131	btLibMeStatusMaxConnections	
0x3132	btLibMeStatusMaxScoConnections	

Code	Identifier	Description
0x3133	btLibMeStatusMaxAclConnections	
0x3134	btLibMeStatusCommandDisallowed	
0x3135	btLibMeStatusLimitedResources	
0x3136	btLibMeStatusSecurityError	
0x3137	btLibMeStatusPersonalDevice	
0x3138	btLibMeStatusHostTimeout	
0x3139	btLibMeStatusUnsupportedFeature	
0x313a	btLibMeStatusInvalidHciParam	
0x313b	btLibMeStatusUserTerminated	
0x313c	btLibMeStatusLowResources	
0x313d	btLibMeStatusPowerOff	
0x313e	btLibMeStatusLocalTerminated	
0x313f	btLibMeStatusRepeatedAttempts	
0x3140	btLibMeStatusPairingNotAllowed	
0x3141	btLibMeStatusUnknownLmpPDU	
0x3142	btLibMeStatusUnsupportedRemote	
0x3143	btLibMeStatusScoOffsetRejected	
0x3144	btLibMeStatusScoIntervalRejected	

Code	Identifier	Description
0x3145	btLibMeStatusScoAirModeRejected	
0x3146	btLibMeStatusInvalidLmpParam	
0x3147	btLibMeStatusUnspecifiedError	
0x3148	btLibMeStatusUnsupportedLmpParam	
0x3149	btLibMeStatusRoleChangeNotAllowed	
0x314a	btLibMeStatusLmpResponseTimeout	
0x314b	btLibMeStatusLmpTransdCollision	
0x314c	btLibMeStatusLmpPduNotAllowed	
0x314d	btLibL2DiscReasonUnknown	
0x314e	btLibL2DiscUserRequest	
0x314f	btLibL2DiscRequestTimeout	
0x3150	btLibL2DiscLinkDisc	
0x3151	btLibL2DiscQosViolation	
0x3152	btLibL2DiscSecurityBlock	
0x3153	btLibL2DiscConnPsmUnsupported	
0x3154	btLibL2DiscConnSecurityBlock	
0x3155	btLibL2DiscConnNoResources	
0x3156	btLibL2DiscConfigUnacceptable	
0x3157	btLibL2DiscConfigReject	

Code	Identifier	Description
0x3158	btLibL2DiscConfigOptions	
0x3159	btLibServiceShutdownAppUse	
0x315a	btLibServiceShutdownPowerCycled	
0x315b	btLibServiceShutdownAclDrop	
0x315c	btLibServiceShutdownTimeout	
0x315d	btLibServiceShutdownDetached	
0x315e	btLibErrInUseByService	
0x315f	btLibErrNoPiconet	
0x3160	btLibErrRoleChange	
0x3161	btLibErrSdpNotMapped	
0x3162	btLibErrAlreadyConnected	
0x3163	btLibErrStackNotOpen	
0x3164	btLibErrBatteryTooLow	
0x3165	btLibErrNotFound	
0x31ef	btLibNotYetSupported	
0x31ff	btLibErrError	
0x3200	udaErrorClass	UDA Manager errors
0x3201	udaErrControl	
0x3800	cpmErrorClass	Cryptographic Provider Manager errors
0x3801	cpmErrAlreadyOpen	
0x3802	cpmErrNotOpen	
0x3803	cpmErrStillOpen	
0x3804	cpmErrNoProviders	

Code	Identifier	Description
0x3805	cpmErrNoBaseProvider	
0x3806	cpmErrProviderNotFound	
0x380a	cpmErrParamErr	
0x380b	cpmErrOutOfResources	
0x380c	cpmErrOutOfMemory	
0x380d	cpmErrBufTooSmall	
0x380e	cpmErrBadData	
0x380f	cpmErrUnimplemented	
0x3810	cpmErrUnsupported	
0x3811	cpmErrNoGlobals	
0x3812	cpmErrKeyExists	
0x3813	cpmErrKeyNotFound	
0x3900	sslErrorClass	SSL errors
0x3900	sslErrOk	Not an error.
0x3901	sslErrFailed	A generic error.
0x3902	sslErrEof	Error returned by SslLib functions when either the SSL protocol has been closed or the underlying socket has been closed. This error indicates that the current SslContext is unable to read or write any more data bytes.
0x3903	sslErrOutOfMemory	Returned if a dynamic memory allocation failed. This is normally considered a very bad error.

Code	Identifier	Description
0x3904	sslErrCbAbort	This error code would be returned by an applications callback function to indicate a desire to exit. This error may not be fatal, depending on the callback that generated the error.
0x3905	sslErrIo	This error code is returned when an underlying Net Library function call returned an error that is not fatal. A timeout, or other such non-fatal network errors will be reclassified as this error type. A function that returns this error type can be re-called once the error condition has disappeared.
0x3906	sslErrNotFound	Returned on an internal SslLib search that did not find a valid entry. Consider this an internal SslLib error.
0x3907	sslErrDivByZero	Something went wrong in the Maths library. These error will normally only be generated by certificates which have invalid public keys.
0x3908	sslErrNoModInverse	Something went wrong in the Maths library. These error will normally only be generated by certificates which have invalid public keys.
0x3909	sslErrBadDecode	An error occurred while decoding values during certificate verification.
0x390a	sslErrInitNotCalled	An internal SslLib error.
0x390b	sslErrBufferTooSmall	A supplied buffer was not large enough for the output data.
0x390c	sslErrNullArg	An passed argument was `NULL` that should not have been `NULL`.
0x390d	sslErrBadLength	A length argument was invalid.

Code	Identifier	Description
0x390e	sslErrNoDmem	An internal SslLib error.
0x390f	sslErrNoMethodSet	An internal SslLib error.
0x3910	sslErrNoRandom	A problem with the random number source.
0x3911	sslErrBadArgument	An invalid argument was provided to the function.
0x3912	sslErrBadOption	An invalid argument was provided to the function.
0x3913	sslErrNotImplemented	An internal SslLib error.
0x3914	sslErrReallocStaticData	An internal SslLib error
0x3915	sslErrInternalError	An internal SslLib error.
0x3925	sslErrRecordError	An invalid record was received in the SslContext.
0x3926	sslErrCsp	A generic error occurred inside the SslLib cryptographic library.
0x3927	sslErrCert	A generic error occurred inside the SslLib certificate library.
0x3928	sslErrHandshakeEncoding	An error occurred during decoding of SSL handshake messages.
0x3929	sslErrMissingProvider	An internal SslLib error.
0x392a	sslErrHandshakeProtocol	An error occurred while processing the decoded SSL handshake messages.
0x392b	sslErrExtraHandshakeData	Extra data was found in the SSL handshake messages that should not have been there.
0x392c	sslErrWrongMessage	An invalid SSL message was received.
0x392d	sslErrFatalAlert	A fatal alert was received by the SSL protocol.

Code	Identifier	Description
0x392e	sslErrBadPeerFinished	The final check of the SSL handshake failed. This indicates that there was a problem establishing a shared secret value. It could be caused by the server using a certificate that does not match its private key.
0x392f	sslErrBadSignature	An invalid signature was found on a ephemeral Cipher Suite message.
0x3931	sslErrUnexpectedRecord	A record of the wrong was received inside the SSL protocol .
0x3932	sslErrReadAppData	Application data was read by the SSL protocol when it was expecting handshake messages.
0x3933	sslErrCertDecodeError	The Servers certificate could not be decoded.
0x3934	sslErrUnsupportedCertType	The Servers certificate contains a public key we cannot decode.
0x3935	sslErrUnsupportedSignatureType	We have been send a certificate with a signature type we do not recognize.
0x3980	sslErrVerifyBadSignature	
0x3981	sslErrVerifyNoTrustedRoot	
0x3982	sslErrVerifyNotAfter	
0x3983	sslErrVerifyNotBefore	
0x3984	sslErrVerifyConstraintViolation	
0x3985	sslErrVerifyUnknownCriticalExtension	
0x3986	sslErrVerifyCallback	
0x3a00	uilibErrorClass	User Interface Library errors

Code	Identifier	Description
0x3a01	uilibErrInvalidParam	At least one of the supplied parameters is invalid.
0x3a02	uilibErrCurrentFocusInvalid	There is no current focus object.
0x3a03	uilibErrObjectFocusModeOff	The specified form is not in object focus mode. For the `FrmGlueNav...` functions in the Palm OS Glue library, this error code is also used to indicate that the device doesn't support one-handed navigation.
0x3a04	uilibErrObjectNotFound	There is no object with a system-drawn focus ring.
0x3a05	uilibErrNoNavInfoForForm	The specified form has no navigation information.
0x3a06	uilibErrInvalidFocusObject	The control to which focus is being transferred is not marked as usable.
0x3a07	uilibErrFormDoesNotHaveFocus	The specified form doesn't currently have the focus.
0x4800	certErrorClass	Certificate Manager errors
0x4801	certMgrErrNotImplemented	The requested certificate format is not supported.
0x4802	certMgrErrInvalidEncoding	The specified format encoding is invalid.
0x4803	certMgrErrOutOfMemory	There was insufficient memory to complete the operation.
0x4804	certMgrErrInvalidParam	One of the function parameters is invalid.
0x4805	certMgrErrServiceNotStarted	The Certificate Manager process has not started.
0x4806	certMgrErrOutOfResources	The Certificate Manager ran out of resources.

Code	Identifier	Description
0x4807	certMgrErrBufTooSmall	The export buffer is too small. The required size is written into the variable pointed to by the length parameter.
0x4808	certMgrErrFieldNotFound	The specified field could not be found.
0x4809	certMgrErrCertNotFound	A certificate matching the specified criteria was not found.
0x480A	certMgrErrNotExportable	The certificate is not exportable. It is probably stored in compressed form.
0x480B	certMgrErrDatabaseFail	A Data Manager error occurred.
0x480C	certMgrErrBackupInProgress	The certificate vault could not be accessed because it is in the process of being backed up.
0x480D	certMgrErrNotRemovable	The certificate is not removable.
0x5000	pinsErrorClass	Pen Input Services errors
0x5001	pinErrNoSoftInputArea	
0x5002	pinErrInvalidParam	
0x5100	statErrorClass	Status Bar Manager errors
0x5101	statErrInvalidLocation	
0x5102	statErrInvalidName	
0x5103	statErrInputWindowOpen	
0x7000	oemErrorClass	OEM/Licensee errors
0x7f00	errInfoClass	Special class shows information without error code.
0x8000	appErrorClass	Application-defined errors

Index

Symbols

() *63*
_searchF 1037

Numerics

2.0 feature set 2382, 2459
3.0 feature set 2385, 2438, 2440
3.1 feature set 2389
3.2 feature set 2392
3.5 feature set 2402, 2421, 2423
4.0 feature set 2409
5.0 feature set 2426
5.1 feature set 2431
5.3SC feature set 2437
5.4 feature set 2438

A

accented characters and StrToLower 1009
active form 299
active window 75, 1233, 1288
adding event to event queue 1015
AddressLookupFields *497*
AddrLookupParamsType *498*
AdnDbgPostLoadParamsType *516*
AdnDbgPostUnloadParamsType *516*
AdnDebugBreak() *519*
AdnDebugEnableGet() *519*
AdnDebugEnableGetSupported() *520*
AdnDebugEnableSet() *520*
AdnDebugLicenseeSpecific() *521*
AdnDebugMessage() *521*
AdnDebugMessageIf() *522*
AdnDebugNativeRegister() *522*
AdnDebugNativeRegisterAddr() *523*
AdnDebugNativeUnregister() *524*
AdnDebugNativeUnregisterAddr() *524*
AdnDebugUpdateLoadedModules() *525*
adnFtrCreator *518*
adnFtrNumVersion *519*
AdnGetNativeCodeBaseAddr() *525*
afterPresetDelay *898*
AlarmMgr.h 509
alarms 509–513
  and launch codes 16
  canceling 510
  procedure alarms 512
  setting 510
  sysAppLaunchCmdTimeChange 36
alerts 290
  custom alert 293, 294
  SysFatalAlert 428
AlertTemplateType *259*
AlertType *279*
alertTypes *279*
allocating chunks on dynamic heap 851
AlmAlarmProcPtr() *513*
almErrFull 511, 512
almErrMemory 511, 512
AlmGetAlarm 509
AlmGetAlarm() *509*
AlmGetProcAlarm() *510*
almProcCmdCustom 514
AlmProcCmdEnum 513
almProcCmdReschedule 513
almProcCmdTriggered 513
AlmSetAlarm 510
AlmSetAlarm() *510*
AlmSetProcAlarm() *512*
alphaGraffitiSilkscreenArea 1024
APAlgorithmEnum 2127
apCertMgrElementDataTypeASN1BitString *580*
apCertMgrElementDataTypeASN1BmpString *580*
apCertMgrElementDataTypeASN1Boolean *580*
apCertMgrElementDataTypeASN1EmbeddedPDV *580*
apCertMgrElementDataTypeASN1Enumerated *580*
apCertMgrElementDataTypeASN1Eoc *580*
apCertMgrElementDataTypeASN1External *580*
apCertMgrElementDataTypeASN1GenString *580*
apCertMgrElementDataTypeASN1GenTime *580*
apCertMgrElementDataTypeASN1GraphicString *580*
apCertMgrElementDataTypeASN1IA5String *580*
apCertMgrElementDataTypeASN1Integer *580*
apCertMgrElementDataTypeASN1ISO64String *580*

apCertMgrElementDataTypeASN1Null *580*
apCertMgrElementDataTypeASN1NumericString *580*
apCertMgrElementDataTypeASN1ObjDesc *580*
apCertMgrElementDataTypeASN1OctetString *580*
apCertMgrElementDataTypeASN1OID *580*
apCertMgrElementDataTypeASN1PrintString *580*
apCertMgrElementDataTypeASN1Real *580*
apCertMgrElementDataTypeASN1Sequence *580*
apCertMgrElementDataTypeASN1Set *580*
apCertMgrElementDataTypeASN1T61String *580*
apCertMgrElementDataTypeASN1UnivString *581*
apCertMgrElementDataTypeASN1UTCTime *581*
apCertMgrElementDataTypeASN1UTF8String *581*
apCertMgrElementDataTypeASN1VideoTexString *581*
apCertMgrElementFieldEntireCert *576*
apCertMgrElementFieldExtension *577*
apCertMgrElementFieldExtensions *577*
apCertMgrElementFieldInnerDER *577*
apCertMgrElementFieldIssuerID *577*
apCertMgrElementFieldIssuerRDN *577*
apCertMgrElementFieldIssuerUniqueID *577*
apCertMgrElementFieldNotAfter *577*
apCertMgrElementFieldNotBefore *577*
apCertMgrElementFieldPubKeyBER *577*
apCertMgrElementFieldRDNOID *579*
apCertMgrElementFieldRDNOIDN() *584*
apCertMgrElementFieldRDNValue *579*
apCertMgrElementFieldRDNValueN() *585*
apCertMgrElementFieldRSAModulus *578*
apCertMgrElementFieldRSAPubExpo *578*
apCertMgrElementFieldSerialNumber *577*
apCertMgrElementFieldSigAlgID *577*
apCertMgrElementFieldSignature *577*
apCertMgrElementFieldSigOID *577*
apCertMgrElementFieldSigParams *577*
apCertMgrElementFieldSubjectID *577*
apCertMgrElementFieldSubjectRDN *577*
apCertMgrElementFieldSubjectUniqueID *577*
apCertMgrElementFieldVersion *577*
apCertMgrElementFieldX509ExBytes *579*
apCertMgrElementFieldX509ExBytesN() *585*
apCertMgrElementFieldX509ExCritical *579*
apCertMgrElementFieldX509ExCriticalN() *585*
apCertMgrElementFieldX509ExOID *579*
apCertMgrElementFieldX509ExOIDN() *586*
apCertMgrElementTypeRDN *572*
apCertMgrElementTypeRSA *572*
apCertMgrElementTypeX509Cert *572*
apCertMgrElementTypeX509Extensions *572*
apCertMgrFieldExtensions *577*
apCertMgrFieldIssuerID *577*
apCertMgrFieldIssuerRDN *577*
apCertMgrFieldIssuerUniqueID *577*
apCertMgrFieldNotAfter *577*
apCertMgrFieldNotBefore *577*
apCertMgrFieldPubKeyBER *578*
apCertMgrFieldSerialNumber *578*
apCertMgrFieldSignature *578*
apCertMgrFieldSigOID *578*
apCertMgrFieldSigParams *578*
apCertMgrFieldSubjectID *578*
apCertMgrFieldSubjectRDN *578*
apCertMgrFieldSubjectUniqueID *578*
apCertMgrFieldVersion *578*
apCertMgrFormatX509 *581*
apCertMgrFormatXML *581*
apCertMgrSearchCert *574*
apCertMgrSearchCertID *574*
apCertMgrSearchSubjectRDN *574*
APCipherInfoStruct *2134*
APHashInfoStruct *2135*
APKeyClassEnum 2130
apKeyClassPrivate *2130*
apKeyClassPublic *2130*
apKeyClassSymmetric *2130*
apKeyClassUnspecified *2130*
APKeyInfoStruct *2136*
apKeyUsageAll *2131*
apKeyUsageCertificateSigning *2131*
apKeyUsageDataEncrypting *2131*
apKeyUsageEncryption *2131*
APKeyUsageEnum 2131
apKeyUsageKeyEncrypting *2131*

apKeyUsageMessageIntegrity *2131*
apKeyUsageSigning *2131*
apKeyUsageUnspecified *2131*
apModeCounter *2126*
APModeEnum 2126
apModeTypeCBC *2126*
apModeTypeCBC_CTS *2126*
apModeTypeCFB *2126*
apModeTypeECB *2126*
apModeTypeNone *2126*
apModeTypeOFB *2126*
apModeTypeUnspecified *2126*
APPaddingEnum 2131
apPaddingTypeNone *2132*
apPaddingTypeOAEP *2132*
apPaddingTypePKCS1Type1 *2132*
apPaddingTypePKCS1Type2 *2132*
apPaddingTypePKCS5 *2132*
apPaddingTypeSSLv23 *2132*
apPaddingTypeUnspecified *2132*
appErrorClass 695
appEvtHookKeyMask 61
AppInfoPtr *145*, 145
appInfoStringsRsc 156
AppInfoType *145*
AppLaunchCmd.h 497
Application focus mode 285
application preferences 912
applications
    Security 35
APProviderContextStruct *2137*
APProviderInfoStruct *2137*
APProviderInfoType 2137
appStopEvent *45*
APVerifyInfoStruct *2138*
archiving
    marking record as archived 612
ARM-native code
    calling 1316
atoi function substitute (StrAToI) 995
atPresetTime *898*
Attention Manager 102, 119
AttnCallbackProc() *141*
AttnCommandArgsType *121*
AttnCommandType *119*
AttnDoSpecialEffects() *131*
attnErrMemory 128
AttnFlagsType *124*
AttnForgetIt() *132*
AttnGetAttention() *133*
AttnGetCounts() *135*
AttnIndicatorEnable() *136*
AttnIndicatorEnabled() *137*
attnIndicatorEnterEvent *46*, 46
attnIndicatorSelectEvent *46*
AttnIterate() *137*
AttnLaunchCodeArgsType *126*
AttnLevelType *127*
AttnListOpen() *138*
AttnNotifyDetailsType 102
AttnUpdate() *139*
auto-off
    setting 1056
    timer 1027
autoRepeatKeyMask 61

B

badDrawWindowValue 1293
BarBeamBitmap 391
BarCopyBitmap 391
BarCutBitmap 391
BarDeleteBitmap 391
BarInfoBitmap 391
BarPasteBitmap 391
BarSecureBitmap 391
BarUndoBitmap 391
base 10 form of floating-point number 760, 2462
battery timeout 1033, 1035
battery voltage warning threshold 1033, 1035
Bitmap.h 527
BitmapCompressionType *527*, 527
BitmapCompressionTypeBest *528*
BitmapCompressionTypeEnd *528*
BitmapCompressionTypeNone *528*
BitmapCompressionTypePackBits *528*
BitmapCompressionTypeRLE *528*
BitmapCompressionTypeScanLine *527*

BitmapDirectInfoType *528*, 528
BitmapFlagsType *529*, 529
BitmapPtr *532*, 532
bitmapRsc 547
bitmaps
    drawing 1236
BitmapType *532*, 532
BitmapTypeV0 *534*
BitmapTypeV1 *535*
BitmapTypeV2 *537*
BitmapTypeV3 *539*
BitmapVersionOne 546
BitmapVersionTwo 546
BitmapVersionZero 546
blackPattern *1218*
blank lines in field 221
Bluetooth Sleep and Wake Scheduling Feature 2439, 2440
BmpBitsSize() *547*
BmpColortableSize() *548*
BmpCompress() *548*
BmpCreate() *549*
BmpCreateBitmapV3() *551*
BmpDelete() *552*
BmpGetBitDepth() *553*
BmpGetBits() *553*
BmpGetColortable() *554*
BmpGetCompressionType() *554*
BmpGetDensity() *555*
BmpGetDimensions() *555*
BmpGetNextBitmap() *556*
BmpGetNextBitmapAnyDensity() *557*
BmpGetSizes() *558*
BmpGetTransparentValue() *559*
BmpGetVersion() *559*
BmpGlueGetBitDepth 554, 1930
BmpGlueGetBits() *1932*
BmpGlueGetCompressionType() *1933*
BmpGlueGetDimensions 556, 1930
BmpGlueGetNextBitmap 557, 1930
BmpGlueGetTransparentValue() *1933*
BmpGlueSetTransparentValue() *1934*
BmpSetDensity() *560*
BmpSetTransparentValue() *560*
BmpSize() *561*
boldButtonFrame *167*
boldFont *773*, 796, 1941
boot, and heap compacting 843
bound of next line for global find 254
btexgFtrCreator 714
btexgFtrNumVersion 714
BtLibAccessibleModeEnum *1988*
btLibActiveMode *2002*
BtLibAddrAToBtd() *1979*
BtLibAddrBtdToA() *1980*
btLibAvailability *2055*
btLibBrowseGroupList *2055*
btLibCachedOnly *2019*
btLibCachedThenRemote *2019*
BtLibCancelInquiry() *2011*
btLibCharSet_Adobe_Standard_Encoding *2044*
btLibCharSet_Adobe_Symbol_Encoding *2044*
btLibCharSet_ANSI_X3_110_1983 *2044*
btLibCharSet_ASMO_449 *2044*
btLibCharSet_Big5 *2044*
btLibCharSet_Big5_HKSCS *2044*
btLibCharSet_BS_4730 *2044*
btLibCharSet_BS_viewdata *2044*
btLibCharSet_CSA_Z243_4_1985_1 *2044*
btLibCharSet_CSA_Z243_4_1985_2 *2044*
btLibCharSet_CSA_Z243_4_1985_gr *2044*
btLibCharSet_CSN_369103 *2044*
btLibCharSet_DEC_MCS *2044*
btLibCharSet_DIN_66003 *2044*
btLibCharSet_dk_us *2044*
btLibCharSet_DS_2089 *2044*
btLibCharSet_EBCDIC_AT_DE *2044*
btLibCharSet_EBCDIC_AT_DE_A *2045*
btLibCharSet_EBCDIC_CA_FR *2045*
btLibCharSet_EBCDIC_DK_NO *2045*
btLibCharSet_EBCDIC_DK_NO_A *2045*
btLibCharSet_EBCDIC_ES *2045*
btLibCharSet_EBCDIC_ES_A *2045*
btLibCharSet_EBCDIC_ES_S *2045*
btLibCharSet_EBCDIC_FI_SE *2045*
btLibCharSet_EBCDIC_FI_SE_A *2045*
btLibCharSet_EBCDIC_FR *2045*

btLibCharSet_EBCDIC_IT *2045*
btLibCharSet_EBCDIC_PT *2045*
btLibCharSet_EBCDIC_UK *2045*
btLibCharSet_EBCDIC_US *2045*
btLibCharSet_ECMA_cyrillic *2045*
btLibCharSet_ES *2045*
btLibCharSet_ES2 *2045*
btLibCharSet_EUC_JP *2045*
btLibCharSet_EUC_KR *2045*
btLibCharSet_Extended_UNIX_Code_Fixed_Widt h_for_Japanese *2045*
btLibCharSet_GB_1988_80 *2045*
btLibCharSet_GB_2312_80 *2045*
btLibCharSet_GB2312 *2045*
btLibCharSet_GOST_19768_74 *2045*
btLibCharSet_greek_ccitt *2045*
btLibCharSet_greek7 *2045*
btLibCharSet_greek7_old *2045*
btLibCharSet_HP_DeskTop *2045*
btLibCharSet_HP_Legal *2046*
btLibCharSet_HP_Math8 *2046*
btLibCharSet_HP_Pi_font *2046*
btLibCharSet_hp_roman8 *2046*
btLibCharSet_HZ_GB_2312 *2046*
btLibCharSet_IBM_Symbols *2048*
btLibCharSet_IBM_Thai *2048*
btLibCharSet_IBM00858 *2047*
btLibCharSet_IBM00924 *2047*
btLibCharSet_IBM01140 *2047*
btLibCharSet_IBM01141 *2047*
btLibCharSet_IBM01142 *2047*
btLibCharSet_IBM01143 *2047*
btLibCharSet_IBM01144 *2047*
btLibCharSet_IBM01145 *2047*
btLibCharSet_IBM01146 *2047*
btLibCharSet_IBM01147 *2048*
btLibCharSet_IBM01148 *2048*
btLibCharSet_IBM01149 *2048*
btLibCharSet_IBM037 *2046*
btLibCharSet_IBM038 *2046*
btLibCharSet_IBM1026 *2047*
btLibCharSet_IBM273 *2046*
btLibCharSet_IBM274 *2046*
btLibCharSet_IBM275 *2046*
btLibCharSet_IBM277 *2046*
btLibCharSet_IBM278 *2046*
btLibCharSet_IBM280 *2046*
btLibCharSet_IBM281 *2046*
btLibCharSet_IBM284 *2046*
btLibCharSet_IBM285 *2046*
btLibCharSet_IBM290 *2046*
btLibCharSet_IBM297 *2046*
btLibCharSet_IBM420 *2046*
btLibCharSet_IBM423 *2046*
btLibCharSet_IBM424 *2046*
btLibCharSet_IBM437 *2046*
btLibCharSet_IBM500 *2046*
btLibCharSet_IBM775 *2046*
btLibCharSet_IBM850 *2046*
btLibCharSet_IBM851 *2046*
btLibCharSet_IBM852 *2046*
btLibCharSet_IBM855 *2046*
btLibCharSet_IBM857 *2047*
btLibCharSet_IBM860 *2047*
btLibCharSet_IBM861 *2047*
btLibCharSet_IBM862 *2047*
btLibCharSet_IBM863 *2047*
btLibCharSet_IBM864 *2047*
btLibCharSet_IBM865 *2047*
btLibCharSet_IBM866 *2047*
btLibCharSet_IBM868 *2047*
btLibCharSet_IBM869 *2047*
btLibCharSet_IBM870 *2047*
btLibCharSet_IBM871 *2047*
btLibCharSet_IBM880 *2047*
btLibCharSet_IBM891 *2047*
btLibCharSet_IBM903 *2047*
btLibCharSet_IBM904 *2047*
btLibCharSet_IBM905 *2047*
btLibCharSet_IBM918 *2047*
btLibCharSet_IEC_P27_1 *2048*
btLibCharSet_INIS *2048*
btLibCharSet_INIS_8 *2048*
btLibCharSet_INIS_cyrillic *2048*
btLibCharSet_INVARIANT *2048*
btLibCharSet_ISO_10367_box *2049*

btLibCharSet_ISO_10646_UCS_2 *2049*
btLibCharSet_ISO_10646_UCS_4 *2049*
btLibCharSet_ISO_10646_UCS_Basic *2049*
btLibCharSet_ISO_10646_Unicode_Latin1 *2049*
btLibCharSet_ISO_10646_UTF_1 *2049*
btLibCharSet_ISO_2022_CN *2048*
btLibCharSet_ISO_2022_CN_EXT *2048*
btLibCharSet_ISO_2022_JP *2048*
btLibCharSet_ISO_2022_JP_2 *2048*
btLibCharSet_ISO_2022_KR *2048*
btLibCharSet_ISO_2033_1983 *2048*
btLibCharSet_ISO_5427 *2048*
btLibCharSet_ISO_5427_1981 *2048*
btLibCharSet_ISO_5428_1980 *2048*
btLibCharSet_ISO_646_basic_1983 *2048*
btLibCharSet_ISO_646_irv_1983 *2048*
btLibCharSet_ISO_6937_2_25 *2048*
btLibCharSet_ISO_6937_2_add *2048*
btLibCharSet_ISO_8859_1 *2048*
btLibCharSet_ISO_8859_1_Windows_3_0_Latin_1 *2049*
btLibCharSet_ISO_8859_1_Windows_3_1_Latin_1 *2049*
btLibCharSet_ISO_8859_10 *2048*
btLibCharSet_iso_8859_13 *2048*
btLibCharSet_iso_8859_14 *2048*
btLibCharSet_ISO_8859_15 *2048*
btLibCharSet_ISO_8859_2 *2049*
btLibCharSet_ISO_8859_2_Windows_Latin_2 *2049*
btLibCharSet_ISO_8859_3 *2049*
btLibCharSet_ISO_8859_4 *2049*
btLibCharSet_ISO_8859_5 *2049*
btLibCharSet_ISO_8859_6 *2049*
btLibCharSet_ISO_8859_6_E *2049*
btLibCharSet_ISO_8859_6_I *2049*
btLibCharSet_ISO_8859_7 *2049*
btLibCharSet_ISO_8859_8 *2049*
btLibCharSet_ISO_8859_8_E *2049*
btLibCharSet_ISO_8859_8_I *2049*
btLibCharSet_ISO_8859_9 *2049*
btLibCharSet_ISO_8859_9_Windows_Latin_5 *2049*
btLibCharSet_ISO_8859_supp *2049*
btLibCharSet_iso_ir_90 *2049*
btLibCharSet_ISO_Unicode_IBM_1261 *2049*
btLibCharSet_ISO_Unicode_IBM_1264 *2049*
btLibCharSet_ISO_Unicode_IBM_1265 *2049*
btLibCharSet_ISO_Unicode_IBM_1268 *2049*
btLibCharSet_ISO_Unicode_IBM_1276 *2050*
btLibCharSet_IT *2050*
btLibCharSet_JIS_C6220_1969_jp *2050*
btLibCharSet_JIS_C6220_1969_ro *2050*
btLibCharSet_JIS_C6226_1978 *2050*
btLibCharSet_JIS_C6226_1983 *2050*
btLibCharSet_JIS_C6229_1984_a *2050*
btLibCharSet_JIS_C6229_1984_b *2050*
btLibCharSet_JIS_C6229_1984_b_add *2050*
btLibCharSet_JIS_C6229_1984_hand *2050*
btLibCharSet_JIS_C6229_1984_hand_add *2050*
btLibCharSet_JIS_C6229_1984_kana *2050*
btLibCharSet_JIS_Encoding *2050*
btLibCharSet_JIS_X0201 *2050*
btLibCharSet_JIS_X0212_1990 *2050*
btLibCharSet_JUS_I_B1_002 *2050*
btLibCharSet_JUS_I_B1_003_mac *2050*
btLibCharSet_JUS_I_B1_003_serb *2050*
btLibCharSet_KOI8_R *2050*
btLibCharSet_KOI8_U *2050*
btLibCharSet_KS_C_5601_1987 *2050*
btLibCharSet_KSC5636 *2050*
btLibCharSet_latin_greek *2050*
btLibCharSet_Latin_greek_1 *2050*
btLibCharSet_latin_lap *2050*
btLibCharSet_macintosh *2050*
btLibCharSet_Microsoft_Publishing *2050*
btLibCharSet_MNEM *2050*
btLibCharSet_MNEMONIC *2051*
btLibCharSet_MSZ_7795_3 *2051*
btLibCharSet_NATS_DANO *2051*
btLibCharSet_NATS_DANO_ADD *2051*
btLibCharSet_NATS_SEFI *2051*
btLibCharSet_NATS_SEFI_ADD *2051*
btLibCharSet_NC_NC00_10_81 *2051*
btLibCharSet_NF_Z_62_010 *2051*
btLibCharSet_NF_Z_62_010__1973_ *2051*
btLibCharSet_NS_4551_1 *2051*

btLibCharSet_NS_4551_2 *2051*
btLibCharSet_PC8_Danish_Norwegian *2051*
btLibCharSet_PC8_Turkish *2051*
btLibCharSet_PT *2051*
btLibCharSet_PT2 *2051*
btLibCharSet_SCSU *2051*
btLibCharSet_SEN_850200_B *2051*
btLibCharSet_SEN_850200_C *2051*
btLibCharSet_Shift_JIS *2051*
btLibCharSet_T_101_G2 *2051*
btLibCharSet_T_61_7bit *2051*
btLibCharSet_T_61_8bit *2051*
btLibCharSet_TIS_620 *2051*
btLibCharSet_UNICODE_1_1 *2051*
btLibCharSet_UNICODE_1_1_UTF_7 *2051*
btLibCharSet_UNKNOWN_8BIT *2051*
btLibCharSet_US_ASCII *2044*
btLibCharSet_us_dk *2051*
btLibCharSet_UTF_16 *2051*
btLibCharSet_UTF_16BE *2052*
btLibCharSet_UTF_16LE *2052*
btLibCharSet_UTF_7 *2052*
btLibCharSet_UTF_8 *2052*
btLibCharSet_Ventura_International *2052*
btLibCharSet_Ventura_Math *2052*
btLibCharSet_Ventura_US *2052*
btLibCharSet_videotex_suppl *2052*
btLibCharSet_VIQR *2052*
btLibCharSet_VISCII *2052*
btLibCharSet_windows_1250 *2052*
btLibCharSet_windows_1251 *2052*
btLibCharSet_windows_1252 *2052*
btLibCharSet_windows_1253 *2052*
btLibCharSet_windows_1254 *2052*
btLibCharSet_windows_1255 *2052*
btLibCharSet_windows_1256 *2052*
btLibCharSet_windows_1257 *2052*
btLibCharSet_windows_1258 *2052*
btLibCharSet_Windows_31J *2052*
BtLibClassOfDeviceType *1988*
btLibClientExecutableUrl *2055*
BtLibClose() *2009*
btLibCOD_Audio *1990*
btLibCOD_Capturing *1989*
btLibCOD_Information *1990*
btLibCOD_LimitedDiscoverableMode *1989*
btLibCOD_Major_Any *1994*
btLibCOD_Major_Audio *1990*
btLibCOD_Major_Computer *1990*
btLibCOD_Major_Imaging *1990*
btLibCOD_Major_Lan_Access_Point *1990*
btLibCOD_Major_Mask *1993*
btLibCOD_Major_Misc *1990*
btLibCOD_Major_Peripheral *1990*
btLibCOD_Major_Phone *1990*
btLibCOD_Major_Unclassified *1990*
btLibCOD_Minor_Any *1994*
btLibCOD_Minor_Audio_Any *1995*
btLibCOD_Minor_Audio_CamCorder *1992*
btLibCOD_Minor_Audio_CarAudio *1992*
btLibCOD_Minor_Audio_GameToy *1992*
btLibCOD_Minor_Audio_HandFree *1992*
btLibCOD_Minor_Audio_HeadPhone *1992*
btLibCOD_Minor_Audio_Headset *1992*
btLibCOD_Minor_Audio_HifiAudio *1992*
btLibCOD_Minor_Audio_LoudSpeaker *1992*
btLibCOD_Minor_Audio_MicroPhone *1992*
btLibCOD_Minor_Audio_PortableAudio *1992*
btLibCOD_Minor_Audio_SetTopBox *1992*
btLibCOD_Minor_Audio_Unclassified *1992*
btLibCOD_Minor_Audio_VCR *1992*
btLibCOD_Minor_Audio_VideoCamera *1992*
btLibCOD_Minor_Audio_VideoConf *1992*
btLibCOD_Minor_Audio_VideoDisplayAndLoud Speaker *1992*
btLibCOD_Minor_Audio_VideoMonitor *1992*
btLibCOD_Minor_Comp_Any *1994*
btLibCOD_Minor_Comp_Desktop *1990*
btLibCOD_Minor_Comp_Handheld *1990*
btLibCOD_Minor_Comp_Laptop *1990*
btLibCOD_Minor_Comp_Palm *1990*
btLibCOD_Minor_Comp_Server *1990*
btLibCOD_Minor_Comp_Unclassified *1990*
btLibCOD_Minor_Imaging_Camera *1993*
btLibCOD_Minor_Imaging_Display *1993*
btLibCOD_Minor_Imaging_Printer *1993*

btLibCOD_Minor_Imaging_Scanner *1993*
btLibCOD_Minor_Imaging_Unclassified *1993*
btLibCOD_Minor_Lan_0 *1991*
btLibCOD_Minor_Lan_17 *1991*
btLibCOD_Minor_Lan_33 *1991*
btLibCOD_Minor_Lan_50 *1991*
btLibCOD_Minor_Lan_67 *1991*
btLibCOD_Minor_Lan_83 *1991*
btLibCOD_Minor_Lan_99 *1991*
btLibCOD_Minor_LAN_Any *1994*
btLibCOD_Minor_Lan_NoService *1991*
btLibCOD_Minor_Mask *1993*
btLibCOD_Minor_Peripheral_CardReader *1993*
btLibCOD_Minor_Peripheral_Combo *1992*
btLibCOD_Minor_Peripheral_DigitizerTablet *1993*
btLibCOD_Minor_Peripheral_GamePad *1993*
btLibCOD_Minor_Peripheral_Joystick *1993*
btLibCOD_Minor_Peripheral_Keyboard *1992*
btLibCOD_Minor_Peripheral_Pointing *1992*
btLibCOD_Minor_Peripheral_RemoteControl *1993*
btLibCOD_Minor_Peripheral_Sensing *1993*
btLibCOD_Minor_Peripheral_Unclassified *1992*
btLibCOD_Minor_Phone_Any *1994*
btLibCOD_Minor_Phone_Cellular *1991*
btLibCOD_Minor_Phone_Cordless *1991*
btLibCOD_Minor_Phone_ISDN *1991*
btLibCOD_Minor_Phone_Modem *1991*
btLibCOD_Minor_Phone_Smart *1991*
btLibCOD_Minor_Phone_Unclassified *1991*
btLibCOD_Networking *1989*
btLibCOD_ObjectTransfer *1990*
btLibCOD_Rendering *1989*
btLibCOD_Service_Mask *1993*
btLibCOD_ServiceAny *1994*
btLibCOD_Telephony *1990*
btLibConnectableOnly *1988*
BtLibConnectionRoleEnum *1995*
BtLibDeviceAddressType *1995*
btLibDiscoverableAndConnectable *1988*
BtLibDiscoverMultipleDevices() *2012*
BtLibDiscoverSingleDevice() *2014*
btLibDocumentationUrl *2055*
btLibFeatureCreator 714
BtLibFriendlyNameType *1996*
BtLibGeneralPreferenceEnum 2016
BtLibGetGeneralPreference() *2016*
BtLibGetNameEnum 2019
BtLibGetRemoteDeviceName() *2018*
BtLibGetSelectedDevices() *2020*
BtLibHandleEvent 2372
BtLibHandleTransportEvent 2372
btLibHoldMode *2002*
btLibIconUrl *2055*
BtLibL2CapHToNL() *1981*
BtLibL2CapHToNS() *1981*
BtLibL2CapNToHL() *1981*
BtLibL2CapNToHS() *1982*
BtLibL2CapPsmType *2038*
btLibLangAbkihazian *2039*
btLibLangAfar *2039*
btLibLangAfrikaans *2039*
btLibLangAlbanian *2043*
btLibLangAmharic *2039*
btLibLangArabic *2039*
btLibLangArmenian *2040*
btLibLangAssamese *2039*
btLibLangAymara *2039*
btLibLangAzerbaijani *2039*
btLibLangBashkir *2039*
btLibLangBasque *2040*
btLibLangBengali *2039*
btLibLangBhutani *2040*
btLibLangBihari *2039*
btLibLangBislama *2039*
btLibLangBreton *2039*
btLibLangBulgarian *2039*
btLibLangBurmese *2042*
btLibLangByelorussian *2039*
btLibLangCambodian *2041*
btLibLangCatalan *2039*
btLibLangChinese *2044*
btLibLangCorsican *2039*
btLibLangCroation *2040*
btLibLangCzech *2040*
btLibLangDanish *2040*

btLibLangDutch *2042*
btLibLangEnglish *2040*
btLibLangEsperanto *2040*
btLibLangEstonian *2040*
btLibLangFaroese *2040*
btLibLangFiji *2040*
btLibLangFinnish *2040*
btLibLangFrench *2040*
btLibLangFrisian *2040*
btLibLangGalician *2040*
btLibLangGeorgian *2041*
btLibLangGerman *2040*
btLibLangGreek *2040*
btLibLangGreenlandic *2041*
btLibLangGuarani *2040*
btLibLangGujarati *2040*
btLibLangHausa *2040*
btLibLangHebrew *2041*
btLibLangHindi *2040*
btLibLangHungarian *2040*
btLibLangIcelandic *2041*
btLibLangIndonesian *2041*
btLibLangInterlingua *2040*
btLibLangInterlingue *2041*
btLibLangInupiak *2041*
btLibLangIrish *2040*
btLibLangItalian *2041*
btLibLangJapanese *2041*
btLibLangJavanese *2041*
btLibLangKannada *2041*
btLibLangKashmiri *2041*
btLibLangKazakh *2041*
btLibLangKinyarwanda *2042*
btLibLangKirghiz *2041*
btLibLangKirundi *2042*
btLibLangKorean *2041*
btLibLangKurdish *2041*
btLibLangLaothian *2041*
btLibLangLatin *2041*
btLibLangLatvian *2041*
btLibLangLingala *2041*
btLibLangLithuanian *2041*
btLibLangMacedonian *2041*
btLibLangMalagasy *2041*
btLibLangMalay *2042*
btLibLangMalayalam *2041*
btLibLangMaltese *2042*
btLibLangMaori *2041*
btLibLangMarathi *2042*
btLibLangMoldavian *2042*
btLibLangMongolian *2041*
btLibLangNaura *2042*
btLibLangNepali *2042*
btLibLangNorwegian *2042*
btLibLangOccitan *2042*
btLibLangOriya *2042*
btLibLangOromo *2042*
btLibLangPashto *2042*
btLibLangPersian *2040*
btLibLangPolish *2042*
btLibLangPortuguese *2042*
btLibLangPunjabi *2042*
btLibLangQuechua *2042*
btLibLangRhaeto_Romance *2042*
btLibLangRomanian *2042*
btLibLangRussian *2042*
btLibLangSamoan *2043*
btLibLangSangho *2042*
btLibLangSanskrit *2042*
btLibLangScotsGaelic *2040*
btLibLangSerbian *2043*
btLibLangSerbo_Croation *2042*
btLibLangSesotho *2043*
btLibLangSetswanna *2043*
btLibLangShona *2043*
btLibLangSindhi *2042*
btLibLangSinghalese *2042*
btLibLangSiswati *2043*
btLibLangSlovak *2042*
btLibLangSlovenian *2043*
btLibLangSomali *2043*
btLibLangSpanish *2040*
btLibLangSundanese *2043*
btLibLangSwahili *2043*
btLibLangSwedish *2043*
btLibLangTagalog *2043*

btLibLangTajik *2043*
btLibLangTamil *2043*
btLibLangTatar *2043*
btLibLangTelugu *2043*
btLibLangThai *2043*
btLibLangTibetan *2039*
btLibLangTigrinya *2043*
btLibLangTonga *2043*
btLibLangTsonga *2043*
btLibLangTurkish *2043*
btLibLangTurkmen *2043*
btLibLangTwi *2043*
btLibLanguageBaseAttributeIdList *2055*
BtLibLanguageBaseTripletType *2038*
btLibLangUkranian *2043*
btLibLangUrdu *2043*
btLibLangUzbek *2043*
btLibLangVietnamese *2043*
btLibLangVolapuk *2044*
btLibLangWelsh *2040*
btLibLangWolof *2044*
btLibLangXhosa *2044*
btLibLangYiddish *2041*
btLibLangYoruba *2044*
btLibLangZulu *2044*
BtLibLinkConnect() *2020*
BtLibLinkDisconnect() *2022*
BtLibLinkGetState() *2023*
BtLibLinkModeEnum 2002
btLibLinkPref_Authenticated *2025*
btLibLinkPref_Encrypted *2025*
btLibLinkPref_LinkRole *2025*
BtLibLinkPrefsEnum 2025
BtLibLinkSetState() *2024*
BtLibManagementCallback() *2035*
btLibManagementEventAccessibilityChange *1997*
btLibManagementEventAclConnectInbound *1998*
btLibManagementEventAclConnectOutbound *1998*
btLibManagementEventAclDisconnect *1999*
btLibManagementEventAuthentication Complete 1999
btLibManagementEventAuthenticationComplete *1999*
btLibManagementEventEncryptionChange *1999*
btLibManagementEventInquiryCanceled *2000*
btLibManagementEventInquiryComplete *2000*
btLibManagementEventInquiryResult *2000*
btLibManagementEventLocalNameChange *2001*
btLibManagementEventModeChange *2001*
btLibManagementEventNameResult *2002*
btLibManagementEventPairingComplete *2003*
btLibManagementEventPasskeyRequest *2003*
btLibManagementEventPasskeyRequestComplete *2004*
btLibManagementEventPasskeyRequested 2003
btLibManagementEventPiconetCreated *2004*
btLibManagementEventPiconetDestroyed *2004*
btLibManagementEventRadioState *2004*
btLibManagementEventRoleChange *2005*
BtLibManagementEventType *1996*
btLibMasterRole *1995*
BtLibMutex 2372
btLibNotAccessible *1988*
BtLibOpen() *2009*
BtLibOpenBackground 2372
btLibParkMode *2002*
BtLibPiconetCreate() *2026*
BtLibPiconetDestroy() *2027*
BtLibPiconetLockInbound() *2028*
BtLibPiconetUnlockInbound() *2029*
btLibPref_CurrentAccessible *2017*
btLibPref_LocalClassOfDevice *2017*
btLibPref_Name *2017*
btLibPref_UnconnectedAccessible *2017*
btLibProfileDescriptorList *2055*
BtLibProfileDescriptorListEntryType *2053*
btLibProtocolDescriptorList *2055*
BtLibProtocolDescriptorListEntryType *2053*
btLibProviderNameOffset *2052*
BtLibRegisterManagementNotification() *2030*
btLibRemoteOnly *2019*
BtLibRfCommHToNL() *1982*
BtLibRfCommHToNS() *1983*
BtLibRfCommNToHL() *1983*
BtLibRfCommNToHS() *1983*
BtLibRfCommServerIdType *2054*

BtLibSdpAttributeDataType *2054*
BtLibSdpAttributeIdType *2055*
BtLibSdpCompareUuids() *2088*
BtLibSdpGetPSMByUuid() *2089*
BtLibSdpGetRawDataElementSize() *2090*
BtLibSdpGetRawDataElementType() *2091*
BtLibSdpGetServerChannelByUuid() *2093*
BtLibSdpHToNL() *1984*
BtLibSdpHToNS() *1984*
BtLibSdpNToHL() *1985*
BtLibSdpNToHS() *1985*
BtLibSdpParseRawDataElement() *2094*
BtLibSdpRecordHandle *2056*
BtLibSdpRemoteServiceRecordHandle *2056*
BtLibSdpServiceRecordCreate() *2095*
BtLibSdpServiceRecordDestroy() *2096*
BtLibSdpServiceRecordGetAttribute() *2097*
BtLibSdpServiceRecordGetNumList Entries 2099
BtLibSdpServiceRecordGetNumListEntries() *2099*
BtLibSdpServiceRecordGetNumLists() *2101*
BtLibSdpServiceRecordGetRawAttribute() *2103*
BtLibSdpServiceRecordGetSizeOfRaw Attribute 2105
BtLibSdpServiceRecordGetSizeOfRawAttribute() *2105*
BtLibSdpServiceRecordGetStringOrURL Length 2106
BtLibSdpServiceRecordGetStringOrUrlLength() *2106*
BtLibSdpServiceRecordMapRemote() *2108*
BtLibSdpServiceRecordSetAttribute() *2109*
BtLibSdpServiceRecordSetAttributesFor Socket 2111
BtLibSdpServiceRecordSetAttributesForSocket() *2111*
BtLibSdpServiceRecordSetRawAttribute() *2112*
BtLibSdpServiceRecordsGetByService Class 2114
BtLibSdpServiceRecordsGetByServiceClass() *2114*
BtLibSdpServiceRecordStartAdvertising() *2116*
BtLibSdpServiceRecordStopAdvertising() *2117*
btLibSdpUUID_PROT_FTP *2059*
btLibSdpUUID_PROT_HTTP *2059*
btLibSdpUUID_PROT_IP *2058*
btLibSdpUUID_PROT_L2CAP *2058*
btLibSdpUUID_PROT_OBEX *2058*
btLibSdpUUID_PROT_RFCOMM *2058*
btLibSdpUUID_PROT_SDP *2058*
btLibSdpUUID_PROT_TCP *2058*
btLibSdpUUID_PROT_TCS_AT *2058*
btLibSdpUUID_PROT_TCS_BIN *2058*
btLibSdpUUID_PROT_UDP *2058*
btLibSdpUUID_PROT_WSP *2059*
btLibSdpUUID_SC_BROWSE_GROUP_DESC *2057*
btLibSdpUUID_SC_CORDLESS_TELEPHONY *2058*
btLibSdpUUID_SC_DIALUP_NETWORKING *2058*
btLibSdpUUID_SC_FAX *2058*
btLibSdpUUID_SC_GENERIC_AUDIO *2058*
btLibSdpUUID_SC_GENERIC_FILE_TRANSFER *2058*
btLibSdpUUID_SC_GENERIC_NETWORKING *2058*
btLibSdpUUID_SC_GENERIC_TELEPHONY *2058*
btLibSdpUUID_SC_HEADSET *2058*
btLibSdpUUID_SC_HEADSET_AUDIO_GATEWAY *2058*
btLibSdpUUID_SC_INTERCOM *2058*
btLibSdpUUID_SC_IRMC_SYNC *2058*
btLibSdpUUID_SC_IRMC_SYNC_COMMAND *2058*
btLibSdpUUID_SC_LAN_ACCESS_PPP *2058*
btLibSdpUUID_SC_OBEX_FILE_TRANSFER *2058*
btLibSdpUUID_SC_OBEX_OBJECT_PUSH *2058*
btLibSdpUUID_SC_PNP_INFO *2058*
btLibSdpUUID_SC_PUBLIC_BROWSE_GROUP *2057*
btLibSdpUUID_SC_SERIAL_PORT *2057*
btLibSdpUUID_SC_SERVICE_DISCOVERY_SERVER *2057*
btLibSdpUUID_SC_WAP *2058*
btLibSdpUUID_SC_WAP_CLIENT *2058*
BtLibSdpUuidInitialize() *2118*
BtLibSdpUuidSizeEnum *2056*
BtLibSdpUuidType *2057*

BtLibSdpVerifyRawDataElement() *2118*
BtLibSecurityFindTrustedDeviceRecord() *1976*
BtLibSecurityGetTrustedDeviceRecordInfo() *1977*
BtLibSecurityNumTrustedDeviceRecords() *1978*
BtLibSecurityRemoveTrustedDevice Record 1979
BtLibSecurityRemoveTrustedDevice() *1979*
btLibServiceClassIdList *2055*
BtLibServiceClose 2372
btLibServiceDescriptionOffset *2052*
btLibServiceId *2055*
BtLibServiceIndicateSessionStart 2372
btLibServiceNameOffset *2052*
BtLibServiceOpen 2372
BtLibServicePlaySound 2372
btLibServiceRecordState *2055*
BtLibSetGeneralPreference() *2031*
btLibSlaveRole *1995*
BtLibSleep 2372
btLibSniffMode *2002*
BtLibSocketAdvanceCredit() *2073*
BtLibSocketCallback() *2120*
BtLibSocketClose() *2074*
BtLibSocketConnect() *2075*
BtLibSocketConnectInfoType 2076
BtLibSocketCreate() *2078*
btLibSocketEventConnectedInbound *2061*
btLibSocketEventConnectedOutbound *2062*
btLibSocketEventConnectRequest *2062*
btLibSocketEventData *2062*
btLibSocketEventDisconnected *2063*
btLibSocketEventSdpGetAttribute *2064*
btLibSocketEventSdpGetNumListEntries *2066*
btLibSocketEventSdpGetNumLists *2067*
btLibSocketEventSdpGetPsmByUuid *2070*
btLibSocketEventSdpGetRawAttribute *2068*
btLibSocketEventSdpGetRawAttributeSize *2069*
btLibSocketEventSdpGetServerChannelBy Uuid 2070
btLibSocketEventSdpGetServerChannelByUuid *2 070*
btLibSocketEventSdpGetStringLen *2065*
btLibSocketEventSdpServiceRecordHandle *2063*
btLibSocketEventSendComplete *2071*
BtLibSocketEventType *2059*
BtLibSocketGetInfo() *2079*
btLibSocketInfo_L2CapChannel *2081*
btLibSocketInfo_L2CapPsm *2081*
btLibSocketInfo_MaxRxSize *2081*
btLibSocketInfo_MaxTxSize *2081*
btLibSocketInfo_Protocol *2081*
btLibSocketInfo_RemoteDeviceAddress *2081*
btLibSocketInfo_RfCommOutstandingCredits *208 1*
btLibSocketInfo_RfCommServerId *2081*
btLibSocketInfo_SdpServiceRecordHandle *2082*
btLibSocketInfo_SendPending *2082*
BtLibSocketInfoEnum 2080
BtLibSocketListen() *2082*
BtLibSocketListenInfoType 2083
BtLibSocketRef *2060*
BtLibSocketRespondToConnection() *2085*
BtLibSocketSend() *2087*
BtLibStartInquiry() *2032*
BtLibStringType *2060*
btLibTimeToLive *2055*
BtLibUnload 2372
BtLibUnregisterManagementNotification() *2034*
BtLibUrlType *2060*
btLibUuidSize128 *2057*
btLibUuidSize16 *2056*
btLibUuidSize32 *2057*
BtLibWake 2372
busy bit 660
buttonCtl *170*
ButtonFrameType *167*, 167
buttons (silk-screened buttons) 60
byteAttrFirst 1067
byteAttrHighLow 1068
byteAttrLast 1067
byteAttrMiddle 1067
byteAttrSingle 1067
byteAttrSingleLow 1068

C

calibrating the pen 895
canceling alarms 510
capsLockMask 61

card number 837
categories, setting label 331
category
    DmSeekRecordInCategory 671
    moving records 645
Category Constants 145
Category Data Structures 145
Category Functions 145
CategoryCreateList 153, 2384
CategoryCreateList() *147*
CategoryCreateListV10() *149*
categoryDefaultEditCategoryString 147, 148, 150, 158, 2408
categoryDefaultEditString 149, 158
CategoryEdit 2384
CategoryEdit() *150*
CategoryEditV10() *152*
CategoryEditV20() *151*
CategoryFind() *152*
CategoryFreeList 2384
CategoryFreeList() *153*
CategoryFreeListV10() *154*
CategoryGetName() *154*
CategoryGetNext() *155*
categoryHideEditCategory 147, 148, 149, 158, 2408
CategoryInitialize 146
CategoryInitialize() *156*
CategorySelect 2384
CategorySelect() *157*
CategorySelectV10() *159*
CategorySetName() *160*
CategorySetTriggerLabel() *160*
CategoryTruncateName() *161*
cCountryName constants 827
CertMgrAddCert() *586*
CertMgrCertChainType *572*
CertMgrCertElementEnum *572*
CertMgrCertFieldEnum *573*
CertMgrCertInfoType *573*
CertMgrCertSearchEnum *574*
CertMgrElementListType *574*
CertMgrElementType *575*
certMgrErrBackupInProgress *581*
certMgrErrBufTooSmall *581*
certMgrErrCertNotFound *582*
certMgrErrDatabaseFail *582*
certMgrErrFieldNotFound *582*
certMgrErrInvalidEncoding *582*
certMgrErrInvalidParam *582*
certMgrErrNotExportable *582*
certMgrErrNotImplemented *582*
certMgrErrNotRemovable *582*
certMgrErrOutOfMemory *582*
certMgrErrOutOfResources *582*
certMgrErrServiceNotStarted *582*
CertMgrExportCert() *588*
CertMgrFindCert() *590*
CertMgrGetField() *591*
CertMgrImportCert() *593*
CertMgrReleaseCertInfo() *595*
CertMgrRemoveCert() *595*
CertMgrServiceName *584*
CertMgrVerifyCert() *596*
CertMgrVerifyFail *583*
CertMgrVerifyFailBasicConstraints *583*
CertMgrVerifyFailCriticalExtension *583*
CertMgrVerifyFailKeyUsage *583*
CertMgrVerifyFailNotAfter *583*
CertMgrVerifyFailNotBefore *583*
CertMgrVerifyFailSelfSigned *583*
CertMgrVerifyFailSignature *583*
CertMgrVerifyFailUnknown *583*
CertMgrVerifyFailUnknownIssuer *583*
CertMgrVerifyFailUnknownSigAlg *584*
CertMgrVerifyFailure() *597*
CertMgrVerifyResultType *576*
character attribute functions 599–602
character encodings 1072, 1092, 1099, 1105
characters
    See also multi-byte characters
    attributes 1070, 1081
    converting 1106
    drawable 1080
    drawing in window 1238
    erasing 1245
    graphic 1075
    inverting 1260
    size 1080

sorting text 602
valid 1080
CharAttr.h 599
charAttr_BB 1065
charAttr_CN 1065
charAttr_DI 1065
charAttr_LO 1065
charAttr_PU 1065
charAttr_SP 1065
charAttr_UP 1066
charAttr_XA 1065
charAttr_XD 1066
charAttr_XS 1065
charAttrAlNum 1070
charAttrAlpha 1070
charAttrCntrl 1070
charAttrDelim 1070
charAttrGraph 1070
charAttrPrint 1070
charAttrSpace 1070
charEncoding... constants 826
charEncodingDestBestFitFlag 729
charEncodingHasDoubleByte 1064
charEncodingHasLigatures 1065
charEncodingOnlySingleByte 1064
charEncodingRightToLeft 1065
CharEncodingType 825, *1061*
charEncodingUnknown 1099
checkboxCtl *170*
checkboxFont 779
checkboxTableItem 434
ChrHorizEllipsis 2391
ChrHorizEllipsis() *599*
ChrIsHardKey() *599*
ChrNumericSpace 2391
ChrNumericSpace() *600*
chunks
card number 837
disposing of chunk 838
heap ID 838, 851
locking 838
size 841
unlocking 842, 854
clipboard 214, 215, 233
Clipboard.h 163
ClipboardAddItem() *164*
ClipboardAppendItem() *165*
clipboardBitmap *163*
ClipboardFormatType *163*, 163
ClipboardGetItem() *166*
clipboardInk *163*
clipboardText *163*
Clipper application 2396
clipping rectangle 1290
closing net library 1472, 1485
CncAddProfile() *1336*
CncDefineParamId 1334
CncDefineParamID() *1338*
CncDeleteProfile() *1339*
CncGetParamType 1334
CncGetParamType() *1340*
CncGetProfileInfo() *1340*
CncGetProfileList() *1342*
CncGetSystemFlagBitnum() *1343*
CncGetTrueParamID() *1343*
CncIsFixedLengthParamType() *1344*
CncIsSystemFlags() *1344*
CncIsSystemRange() *1345*
CncIsThirdPartiesRange() *1345*
CncIsVariableLengthParamType() *1346*
cncNotifyProfileEvent *84*, 84
CncProfileCloseDB() *1346*
CncProfileCount() *1347*
CncProfileCreate 1337
CncProfileCreate() *1347*
CncProfileDelete() *1348*
CncProfileGetCurrent() *1348*
CncProfileGetIDFromIndex 1323
CncProfileGetIDFromIndex() *1349*
CncProfileGetIDFromName 1323
CncProfileGetIDFromName() *1350*
CncProfileGetIndex() *1350*
CncProfileID *1323*, 1323
CncProfileNotifyDetailsType 84
CncProfileOpenDB() *1351*
CncProfileSetCurrent() *1352*
CncProfileSettingGet 1332, 1335

CncProfileSettingGet() *1352*
CncProfileSettingSet 1332
CncProfileSettingSet() *1353*
ColorTableEntries() *562*
ColorTableType *542*, 542
colorUnderline *1219*
Command shortcuts 66
commandChr 388, 397, 398, 399, 401
commandKeyMask 61
compacting heaps 843
comparing memory blocks 836
compatibility 2381–2437
confirmationAlert *280*
connect 1552
connection manager 2400
ConnectionMgr.h 1323
constant
    soft 503
constantRscType 503
ContrastAdjust 494, 2405
control objects
    and pen tracking 69
    drawing 176
    erasing 177
    selection in a group 299
    structure 167
Control.h 167
ControlAttrType *168*, 168
controlKeyMask 61
ControlPtr *169*, 169
ControlStyleType *169*, 169, 1934
ControlType 47, 48, *171*, 171
coordinates, display-relative vs. window-relative 1235
CoreTraps.h 1046, 1056
CountryType *823*, 823, 869
CPM and AP Constants 2124
cpmErrAlreadyOpen *2132*
cpmErrBadData *2133*
cpmErrBufTooSmall *2133*
cpmErrKeyExists *2134*
cpmErrKeyNotFound *2134*
cpmErrNoBaseProvider *2133*
cpmErrNoGlobals *2133*
cpmErrNoProviders *2132*
cpmErrNotOpen *2132*
cpmErrOutOfMemory *2133*
cpmErrParamErr *2133*
cpmErrProviderNotFound *2133*
cpmErrStillOpen *2132*
cpmErrUnimplemented *2133*
cpmErrUnsupported *2133*
CPMInfoStruct *2139*
CPMLibDecrypt() *2140*
CPMLibDecryptFinal() *2141*
CPMLibDecryptInit() *2143*
CPMLibDecryptUpdate() *2144*
CPMLibEncrypt() *2144*
CPMLibEncryptFinal() *2146*
CPMLibEncryptInit() *2147*
CPMLibEncryptUpdate() *2148*
CPMLibExportCipherInfo() *2149*
CPMLibExportHashInfo() *2150*
CPMLibExportKeyInfo() *2151*
CPMLibExportVerifyInfo() *2152*
CPMLibGenerateKey 2154
CPMLibGenerateKey() *2153*
CPMLibGetInfo() *2154*
CPMLibGetProviderInfo() *2154*
CPMLibHash() *2155*
CPMLibHashFinal() *2156*
CPMLibHashInit() *2157*
CPMLibHashUpdate() *2158*
CPMLibImportCipherInfo() *2159*
CPMLibImportHashInfo() *2160*
CPMLibImportKeyInfo() *2161*
CPMLibImportVerifyInfo() *2162*
CPMLibReleaseCipherInfo() *2162*
CPMLibReleaseHashInfo() *2163*
CPMLibReleaseKeyInfo() *2163*
CPMLibReleaseVerifyInfo() *2164*
CPMLibVerify() *2164*
CPMLibVerifyFinal() *2166*
CPMLibVerifyInit() *2167*
CPMLibVerifyUpdate() *2168*
Crc.h 1307
Crc16CalcBlock 1307

Crc16CalcBlock() *1307*
creating active window 1233
creating modal window 1233
creator ID 30
CtlDrawControl 168
CtlDrawControl() *176*
CtlEnabled 168
CtlEnabled() *177*
ctlEnterEvent *46*, 47, 48, 180, 313
CtlEraseControl 168
CtlEraseControl() *177*
ctlExitEvent *47*, 48, 180
CtlGetLabel 171
CtlGetLabel() *178*
CtlGetSliderValues() *178*
CtlGetValue 169, 175
CtlGetValue() *179*
CtlGlueGetControlStyle() *1934*
CtlGlueGetFont() *1935*
CtlGlueGetGraphics() *1935*
CtlGlueIsGraphical() *1936*
CtlGlueNewSliderControl() *1936*
CtlGlueSetFont() *1938*
CtlGlueSetFrameStyle() *1938*
CtlGlueSetLeftAnchor() *1939*
CtlHandleEvent 46, 47, 48
CtlHandleEvent() *180*
CtlHideControl 168
CtlHideControl() *181*
CtlHitControl() *181*
CtlNewControl() *182*
CtlNewGraphicControl 182
CtlNewGraphicControl() *184*
CtlNewSliderControl 182
CtlNewSliderControl() *185*
ctlRepeatEvent *47*, 180, 313, 2406
ctlSelectEvent 47, *48*, 313, 2406
CtlSetEnabled 168
CtlSetEnabled() *187*
CtlSetGraphics 173
CtlSetGraphics() *187*
CtlSetLabel 171
CtlSetLabel() *188*
CtlSetSliderValues() *189*
CtlSetUsable 168
CtlSetUsable() *190*
CtlSetValue 169, 175
CtlSetValue() *190*
CtlShowControl 168
CtlShowControl() *191*
CtlValidatePointer() *192*
current time 36
custom fill patterns, getting 1254
customPattern *1218*
CustomPatternType *1211*, 1211
customTableItem 434, 436, 480

D

darkGrayPattern *1218*
data manager error codes 608–611, 636
data storage heap 850
    handles 837
database ID 629
databases
    closing 616
    creating 617
    cutting and pasting 614
    deleting. *See also* DmDatabaseProtect
    overlays 654
    SysCreateDataBaseList 1038
DataMgr.h 603
date 198
Date and Time UI Objects 353
date system resource 195
DateAdjust 1125
DateAdjust() *1125*
DateDaysToDate 1126
DateDaysToDate() *1126*
DateFormatType *1115*
DateGlueTemplateToAscii 1130, 1930
DateGlueToDOWDMFormat 1133, 1930
DatePtr 1119
DateSecondsToDate 1126
DateSecondsToDate() *1126*
dateStringLength 1131
dateTableItem 435
DateTemplateToAscii() *1127*

DateTime.h 1115
DateTimePtr 1118
DateTimeType *1118*
DateToAscii 1131
DateToAscii() *1131*
DateToDays 1132
DateToDays() *1132*
DateToDOWDMFormat 1132
DateToDOWDMFormat() *1132*
DateType *1119*, 1119
day selector object 49
Day.h 195
DayDrawDays() *196*
DayDrawDaySelector() *197*
DayHandleEvent() *197*
DaylightSavingsTypes *1119*
DayOfMonth 1120, 1134
DayOfMonth() *1133*
DayOfMonthType *1120*
DayOfWeek 1134
DayOfWeek() *1134*
DayOfWeekType 1123
daySelectEvent *49*
DaySelectorType *195*
DaysInMonth 1135
DaysInMonth() *1135*
DbgBreak() *687*
DbgCommSettings() *687*
DbgControlFuncType() *690*
dbgCtlAllHandlersID *685*
DbgCtlEnumCallbackFunc() *691*
DbgCtlEnumInfoType *683*
dbgCtlFirstCustomOp *685*
dbgCtlHandled *685*
DbgCtlHandlerInfoType *684*
dbgCtlHandlerNameLen *685*
dbgCtlHandlerVerLen *685*
dbgCtlNotHandled *685*
dbgCtlOpDisableHandler *686*
dbgCtlOpEnableHandler *686*
dbgCtlOpEnumHandlers *686*
dbgCtlOpGetEnabledStatus *686*
dbgCtlOpGetHandlerInfo *686*
dbgCtlOpGetVersion *686*
dbgCtlOpLAST *686*
DbgGetMessage() *688*
DbgInit() *688*
DbgMessage() *689*
DbgSrcBreak() *689*
DbgSrcMessage() *689*
debugging and MemHeapScramble 845
debugging mode 837, 855
defaultAlarmSoundLevel 910
defaultAlarmSoundVolume 910
defaultAutoLockTime 910
defaultAutoLockTimeFlag 910
defaultAutoLockType 910
defaultAutoOffDuration 900, 910
defaultAutoOffDurationSecs 905, 910
defaultBoldFont 1940
defaultGameSoundLevel 910
defaultGameSoundVolume 910
defaultLargeFont 1940
defaultSmallFont 1940
defaultSysSoundLevel 911
defaultSysSoundVolume 911
defaultSystemFont 1940
delete bit 624, 626
delete callback function 1405
DeleteProc 1371, 1373, 1405
deleting databases. *See also* DmDatabaseProtect
deleting records 626
DensityType *543*
DeviceInfoType *1593*, 1614
DeviceInfoType structure 1593
dfDMYLong *1117*
dfDMYLongNoDay *1117*
dfDMYLongWithComma *1117*
dfDMYLongWithDot *1117*
dfDMYWithDashes *1116*
dfDMYWithDots *1116*
dfDMYWithSlashes *1116*
dfMDYLongWithComma *1117*
dfMDYWithDashes *1118*
dfMDYWithSlashes *1116*
dfMYMed *1117*

dfMYMedNoPost *1118*
dfYMDLongWithDot *1117*
dfYMDLongWithSpace *1117*
dfYMDWithDashes *1116*
dfYMDWithDots *1116*
dfYMDWithSlashes *1116*
dialogs
    Edit Categories 150
digitizer
    and PenResetCalibration function 896
    and penUpEvent 69
    EvtProcessSoftKeyStroke 1026
DirectionType 235
DlkCallAppReplyParamType 1309
DlkControl() *1308*
DlkGetSyncInfo() *1311*
dlkSyncStateAborted *1313*
dlkSyncStateCompleted *1313*
dlkSyncStateIncompatibleProducts *1313*
dlkSyncStateInProgress *1313*
dlkSyncStateLocalCan *1313*
dlkSyncStateLostConnection *1313*
dlkSyncStateLowMemoryOnTD *1313*
dlkSyncStateNeverSynced *1313*
dlkSyncStateNPOD *1313*
dlkSyncStateRemoteCan *1313*
DLServer.h 1307
dmAllCategories 153, 605, 652
dmAllHdrAttrs 606
dmAllRecAttrs 605
DmArchiveRecord 612
DmArchiveRecord() *612*
DmAttachRecord 613
DmAttachRecord() *613*
DmAttachResource 615
DmAttachResource() *615*
dmCategoryLength 146, 155, 605
DmCloseDatabase 616
DmCloseDatabase() *616*
DmComparF 632, 643, 662
DmComparF() *680*
DmCreateDatabase 617
DmCreateDatabase() *617*
DmCreateDatabaseFromImage() *619*
DmDatabaseInfo 619
DmDatabaseInfo() *619*
DmDatabaseProtect 616, 621
DmDatabaseProtect() *621*
DmDatabaseSize 622
DmDatabaseSize() *622*
dmDBNameLength 612, 617, 673
DmDeleteCategory 623
DmDeleteCategory() *623*
DmDeleteDatabase 616, 625
DmDeleteDatabase() *625*
DmDeleteRecord 626
DmDeleteRecord() *626*
DmDetachRecord 627
DmDetachRecord() *627*
DmDetachResource 628
DmDetachResource() *628*
dmErrDatabaseNotProtected 609
dmErrRecordArchived 610
DmFindDatabase 618, 625, 629
DmFindDatabase() *629*
DmFindRecordByID 629
DmFindRecordByID() *629*
DmFindResource 630
DmFindResource() *630*
DmFindResourceType 631
DmFindResourceType() *631*
DmFindSortPosition 632, 2384
DmFindSortPosition() *632*
DmFindSortPositionV10() *633*
DmGet1Resource 642, 643, 650
DmGet1Resource() *643*
DmGetAppInfoID 634
DmGetAppInfoID() *634*
DmGetDatabase 625, 635
DmGetDatabase() *635*
DmGetDatabaseLockState() *635*
DmGetLastErr 636
DmGetLastErr() *636*
DmGetNextDatabaseByTypeCreator 637
DmGetNextDatabaseByTypeCreator() *637*
DmGetRecord 640
DmGetRecord() *640*

DmGetResource 641, 642
DmGetResource() *641*
DmGetResourceIndex 642
DmGetResourceIndex() *642*
dmHdrAttrAppInfoDirty 606
dmHdrAttrBackup 606
dmHdrAttrBundle 606
dmHdrAttrCopyPrevention 606
dmHdrAttrHidden 607
dmHdrAttrLaunchableData 607
dmHdrAttrOKToInstallNewer 607
dmHdrAttrOpen 607
dmHdrAttrReadOnly 607
dmHdrAttrRecyclable 616
dmHdrAttrResDB 607
dmHdrAttrResetAfterInstall 607
dmHdrAttrStream 608
DmInsertionSort 643
DmInsertionSort() *643*
dmMaxRecordIndex 605, 613, 648
dmModeExclusive 612
dmModeLeaveOpen 611
dmModeReadOnly 611
dmModeReadWrite 611
dmModeShowSecret 612
dmModeWrite 611
DmMoveCategory 645
DmMoveCategory() *645*
DmMoveRecord 646
DmMoveRecord() *646*
DmNewHandle 647
DmNewHandle() *647*
DmNewRecord 648
DmNewRecord() *648*
DmNewResource 649
DmNewResource() *649*
DmNextOpenDatabase 650
DmNextOpenDatabase() *650*
DmNextOpenResDatabase 650
DmNextOpenResDatabase() *650*
DmNumDatabases 651
DmNumDatabases() *651*
DmNumRecords 651
DmNumRecords() *651*
DmNumRecordsInCategory 652
DmNumRecordsInCategory() *652*
DmNumResources 653
DmNumResources() *653*
DmOpenDatabase 611, 653
DmOpenDatabase() *653*
DmOpenDatabaseByTypeCreator 656
DmOpenDatabaseByTypeCreator() *656*
DmOpenDatabaseInfo 657
DmOpenDatabaseInfo() *657*
DmOpenDBNoOverlay 611, 658
DmOpenDBNoOverlay() *658*
DmOpenRef *603*, 603
DmPositionInCategory 659
DmPositionInCategory() *659*
DmQueryNextInCategory 660
DmQueryNextInCategory() *660*
DmQueryRecord 661
DmQueryRecord() *661*
DmQuickSort 662
DmQuickSort() *662*
dmRecAttrBusy 605
dmRecAttrCategoryMask 605, 645
dmRecAttrDelete 605
dmRecAttrDirty 605
dmRecAttrSecret 606
dmRecNumCategories 146, 605
DmRecordInfo 663
DmRecordInfo() *663*
DmReleaseRecord 641, 648, 664
DmReleaseRecord() *664*
DmReleaseResource 649, 665
DmReleaseResource() *665*
DmRemoveRecord 665
DmRemoveRecord() *665*
DmRemoveResource 666
DmRemoveResource() *666*
DmRemoveSecretRecords 667
DmRemoveSecretRecords() *667*
DmResetRecordStates() *667*
DmResID *603*, 603
DmResizeRecord 667

DmResizeRecord() *667*
DmResizeResource 668
DmResizeResource() *668*
DmResourceInfo 669
DmResourceInfo() *669*
DmResType *604*, 604
DmSearchRecord 670
DmSearchRecord() *670*
DmSearchResource 642, 670
DmSearchResource() *670*
DmSearchStatePtr 638
DmSearchStateType 638
dmSeekBackward 671
dmseekForward 671
DmSeekRecordInCategory 671
DmSeekRecordInCategory() *671*
DmSet 672
DmSet() *672*
DmSetDatabaseInfo 673
DmSetDatabaseInfo() *673*
DmSetRecordInfo 675
DmSetRecordInfo() *675*
DmSetResourceInfo 676
DmSetResourceInfo() *676*
DmStrCopy 677
DmStrCopy() *677*
DmSync() *678*
DmSyncDatabase() *678*
dmSysOnlyHdrAttrs 608
dmSysOnlyRecAttrs 606
dmUnfiledCategory 605, 1378
DmWrite 679
DmWrite() *679*
DmWriteCheck 680
DmWriteCheck() *680*
dom1stFri *1121*
dom1stMon *1121*
dom1stSat *1121*
dom1stSun *1121*
dom1stThu *1121*
dom1stTue *1121*
dom1stWen *1121*
dom2ndFri *1122*
dom2ndMon *1121*
dom2ndSat *1122*
dom2ndSun *1121*
dom2ndThu *1122*
dom2ndTue *1121*
dom2ndWen *1121*
dom3rdFri *1122*
dom3rdMon *1122*
dom3rdSat *1122*
dom3rdSun *1122*
dom3rdThu *1122*
dom3rdTue *1122*
dom3rdWen *1122*
dom4thFri *1122*
dom4thMon *1122*
dom4thSat *1122*
dom4thSun *1122*
dom4thThu *1122*
dom4thTue *1122*
dom4thWen *1122*
domLastFri *1123*
domLastMon *1123*
domLastSat *1123*
domLastSun *1123*
domLastThu *1123*
domLastTue *1123*
domLastWen *1123*
doubleTapKeyMask 61
dowDateStringLength 1133
dowLongDateStrLength 1133
doze mode
    SysTaskDelay 1058
Dragonball EZ 2391
draw window 1293
drawable characters 1080
drawDetail structure 121
drawing rectangular frame 1240, 1243, 1269
drawItemsCallback 361, 371
drawList structure 122
DrawStateFlagsType *1211*
DrawStateType *1212*, 1212
DrvEntryPoint
    for virtual driver 1581
DrvEntryPointProcPtr() *1581*

DrvrInfoType *1567*
DrvrInfoType structure 1567
DrvrRcvQType *1568*
DrvrRcvQType structure 1568
drvrStatusBreakAsserted *1570*
drvrStatusCtsOn *1570*
drvrStatusDataReady *1570*
drvrStatusDsrOn *1570*
DrvrStatusEnum *1570*, 1570
drvrStatusLineErr *1570*
drvrStatusRtsOn *1570*
drvrStatusTxFifoEmpty *1570*
drvrStatusTxFifoFull *1570*
dsAustralia *1120*
dsAustraliaShifted *1120*
dsEasternEuropean *1120*
dsGreatBritain *1120*
dsMiddleEuropean *1120*
dsNone *1120*
dsRumania *1120*
dsTurkey *1120*
dsUSA *1120*
dsWesternEuropean *1120*
dynamic heap
    adding chunk 839
    allocating chunk 851
    moving memory 848
    reinitializing 1055
    test 843
dynamic heap handles 837
dynamic scrolling 72

E

ECFrmValidatePtr() *290*
Edit Categories dialog 150
editingStrID 148, 157
enabling windows 1234
EndianSwap16() *567*
EndianSwap32() *567*
erasing characters 1245
erasing lines in window 1246
erasing rectangle 1247
ErrAlert() *694*
ErrCatch 699
ErrCatch() *695*
ErrDisplay() *696*
ErrDisplayFileLineMsg 697
ErrDisplayFileLineMsg() *697*
ErrEndCatch 699
ErrEndCatch() *697*
ErrExceptionList() *698*
ErrExceptionType *694*
ErrFatalDisplayIf() *698*
errNone 1377, 1626
ErrNonFatalDisplayIf() *699*
error code from data manager call 636
error manager 693–699
ERROR_CHECK_FULL 693
ERROR_CHECK_LEVEL 693, 696, 698, 699
ERROR_CHECK_NONE 693
ERROR_CHECK_PARTIAL 693
errorAlert *280*
ErrorBase.h 693
ErrorMgr.h 693
ErrThrow 699
ErrThrow() *699*
ErrTry 699
ErrTry() *700*
event queue, adding event 1015
Event.H 1015
Event.h 41, 2406
EventPtr *45*, 45
events 41, 76
eventsEnum *42*, 42
EventType 41–??, *43*, ??–76, 2406
EvtAddEventToQueue() *1015*
EvtAddUniqueEventToQueue() *1016*
EvtCopyEvent() *1017*
EvtDequeuePenPoint() *1017*
EvtDequeuePenStrokeInfo() *1018*
EvtEnableGraffiti() *1018*
EvtEnqueueKey() *1019*
EvtEventAvail() *1019*
EvtFlushKeyQueue() *1020*
EvtFlushNextPenStroke() *1020*
EvtFlushPenQueue() *1020*
EvtGetEvent 68, 862, 863

EvtGetEvent() *1021*
EvtGetPen() *1021*
EvtGetPenBtnList() *1022*
EvtGetPenNative() *1023*
EvtGetSilkscreenAreaList() *1024*
EvtKeydownIsVirtual() *1025*
EvtKeyQueueEmpty() *1025*
EvtKeyQueueSize() *1026*
EvtPenQueueSize() *1026*
EvtProcessSoftKeyStroke() *1026*
EvtResetAutoOffTimer 1027
EvtResetAutoOffTimer() *1027*
EvtSetAutoOffCmd 1028
EvtSetAutoOffTimer() *1028*
EvtSetNullEventTick() *1029*
EvtSysEventAvail() *1029*
evtWaitForever 68, 1021, 1686, 1687
EvtWakeup 1030, 1631
EvtWakeup() *1030*
EvtWakeupWithoutNilEvent() *1030*
exchange manager 1355, 2387
ExgAccept 21, 22, 1372, 1375, 1395
ExgAccept() *1366*
exgAskCancel 17, *1356*
exgAskDialog *1355*
exgAskOk 17, *1355*
ExgAskParamType 18
ExgAskResultType *1355*, 1355
exgBeamPrefix 1365
exgBeamScheme 1364, 1365
ExgConnect 1375
ExgConnect() *1367*
ExgControl() *1369*
ExgDBDeleteProcPtr 707, 1186, 1371
ExgDBDeleteProcPtr() *1405*
ExgDBRead 1406
ExgDBRead() *1371*
ExgDBReadProcPtr() *1406*
ExgDBWrite() *1373*
ExgDBWriteProcPtr 1193, 1373
ExgDBWriteProcPtr() *1407*
ExgDialogInfoType 1377, 1378
ExgDisconnect 19, 21, 1368, 1381, 1393, 1395, 1402
ExgDisconnect() *1375*
ExgDoDialog 18, 1355, 1389, 1391
ExgDoDialog() *1377*
exgErrNoKnownTarget 1382, 1386, 1387, 1389, 1391, 1392, 1403
exgErrNotSupported 1359, 1389
ExgGet 1368, 1375, 1381, 1395
exgGet 1391
ExgGet() *1380*
ExgGetDefaultApplication() *1381*
ExgGetRegisteredApplications() *1383*
ExgGetRegisteredTypes() *1384*
ExgGetTargetApplication 1389, 1391
ExgGetTargetApplication() *1385*
ExgGoToType *1356*
ExgLibAccept() *1409*
exgLibAPIVersion 1370
ExgLibClose() *1410*
ExgLibConnect() *1410*
ExgLibControl 1412
ExgLibControl() *1412*
exgLibCtlGetPreview 1370
exgLibCtlGetTitle 1370
exgLibCtlGetVersion 1370
exgLibCtlSpecificOp 1371
ExgLibDisconnect 1413
ExgLibDisconnect() *1413*
ExgLibGet 1414
ExgLibGet() *1414*
ExgLibHandleEvent 1415
ExgLibHandleEvent() *1415*
ExgLibOpen() *1416*
ExgLibPut 1416
ExgLibPut() *1416*
ExgLibReceive 1418
ExgLibReceive() *1418*
ExgLibRequest 1419
ExgLibRequest() *1419*
ExgLibSend 1420
ExgLibSend() *1420*
ExgLibSleep 1421
ExgLibSleep() *1421*
exgLibSmsIncompleteDeleteOp 2183

exgLibSmsIncompleteGetCountOp 2183
exgLibSmsPrefDisplayOp 2183
exgLibSmsPrefGetDefaultOp 2183
exgLibSmsPrefGetOp 2183
exgLibSmsPrefSetOp 2183
ExgLibWake 1422
ExgLibWake() *1422*
ExgLocalLib.h 1355
exgLocalOpAccept 1358
exgLocalOpGet 1358
exgLocalOpGetSender 1358
exgLocalOpNone 1358
exgLocalOpPut 1358
ExgLocalOpType 1357
exgLocalPrefix 1362, 1365
exgLocalScheme 1364, 1365
ExgLocalSocketInfoType *1356*, 1356
exgMaxDescriptionLength 1383, 1396
exgMaxTitleLen 1383
exgMaxTypeLength 1384, 1385, 1396
exgMemError 1385, 1397
ExgMgr.h 701, 1355
exgNoAsk 1390
ExgNotifyGoto 1376, 1392
ExgNotifyGoto() *1388*
ExgNotifyPreview 1358, 1377
ExgNotifyPreview() *1389*
ExgNotifyReceive 1387, 1388
ExgNotifyReceive() *1390*
exgPreviewDialog 20, 1359
exgPreviewDraw 20, 1359
exgPreviewFirstUser 21, 1360
ExgPreviewInfoType 1357, *1358*, 1358, 1389
exgPreviewLastUser 21, 1360
exgPreviewLongString 20, 1359
exgPreviewQuery 20, 1359
exgPreviewShortString 1359
ExgPut 19, 1368, 1369, 1374, 1375, 1402
ExgPut() *1393*
ExgReceive 21, 22, 1368, 1372, 1381, 1406
ExgReceive() *1394*
exgRegCreatorID 1363, 1364, 1398
exgRegExtensionID 1364, 1382
ExgRegisterData 28
ExgRegisterData() *1400*
ExgRegisterDatatype() *1396*
ExgRegisterDataV35 1400
exgRegSchemeID 1363, 1364, 1382, 1399
exgRegTypeID 1364, 1398
ExgRequest 27
ExgRequest() *1401*
ExgSend 19, 1368, 1374, 1393, 1407
ExgSend() *1402*
exgSendBeamPrefix 1365
exgSendPrefix 1365
exgSendScheme 1364, 1365, 1398
ExgSetDefaultApplication 1398, 1399
ExgSetDefaultApplication() *1403*
ExgSocketType 19, 705, 1356, *1360*, 1371, 1373
exgTitleBufferSize 1370
exgUnwrap 1390, 1392, 1396
Expansion Manager 89, 90, 701
expCapabilityHasStorage 702
expCapabilityReadOnly 702
expCapabilitySerial 702
ExpCardGetSerialPort() *705*
ExpCardInfo() *706*
ExpCardInfoType *701*
ExpCardPresent() *706*
expErrCardBadSector 703
expErrCardNoSectorReadWrite 703
expErrCardNotPresent 703
expErrCardProtectedSector 703
expErrCardReadOnly 703
expErrEnumerationEmpty 703
expErrIncompatibleAPIVer 703
expErrInvalidSlotRefNum 703
expErrNotEnoughPower 703
expErrNotOpen 703
expErrSlotDeallocated 703
expErrStillOpen 703
expErrUnimplemented 703
expErrUnsupportedOperation 703
expFtrIDVersion 717
expHandledSound 90
expHandledVolume 90

expMediaType_Any 704
expMediaType_CompactFlash 704
expMediaType_MacSim 704
expMediaType_MemoryStick 704
expMediaType_MultiMediaCard 704
expMediaType_PoserHost 704
expMediaType_RAMDisk 704
expMediaType_SecureDigital 704
expMediaType_SmartMedia 704
ExpSlotDriverInstall() *707*
ExpSlotDriverRemove() *708*
ExpSlotEnumerate() *709*
ExpSlotLibFind() *710*
extended font resource (nfnt) 782
extended gadget 264, 336, 346
extern *685*
EZ Dragonball 2391

F

fatal alert 428
FatalAlert.h 427
fcntl 1547
FeatureMgr.h 713
features. *See* functions starting with Ftr
feedbackSliderCtl *170*
fgetc() *977*
fgets() *978*
field objects
    and text height 225
    modifying 216
    structure 208
Field.h 205
FieldAttrType *205*, 205
FieldPtr *208*, 208
FieldType *208*, 208
file mode constants *737*, 738
file streaming 2387
FileClearerr() *738*
FileClose() *739*
FileControl() *739*
FileDelete() *745*
FileDmRead() *745*
FileEOF() *746*
FileError() *747*
FileFlush() *747*
FileGetLastError() *748*
FileInfoType *1143*
FileOpen() *748*
FileOpEnum 740
FileOriginEnum 752
FileRead() *750*
FileRef *1144*
FileRewind() *751*
FileSeek() *752*
FileStream.h 737
FileTell() *753*
FileTruncate() *753*
FileWrite() *754*
fill patterns
    getting 1254
    setting 1295
Find (global find) 22, 25, 253–257
    saving data 34
Find (lookup) 31
Find icon 60
Find.h 253
FindDrawHeader 24, 253
FindDrawHeader() *253*
FindGetLineBounds 254
FindGetLineBounds() *254*
FindParamsType 22, 23
FindSaveMatch 24, 27, 254
FindSaveMatch() *254*
FindStrInStr 24, 256
FindStrInStr() *256*
flags, launch flags 37
FldCalcFieldHeight() *213*
fldChangedEvent *50*, 236, 2385
FldCompactText() *214*
FldCopy() *214*
FldCut() *215*
FldDelete() *215*
FldDirty() *216*
FldDrawField() *217*
fldEnterEvent 50, *50*, 227, 313
FldEraseField() *217*
FldFreeMemory() *218*

FldGetAttributes() *218*
FldGetBounds() *219*
FldGetFont() *219*
FldGetInsPtPosition() *220*
FldGetMaxChars() *220*
FldGetNumberOfBlankLines() *221*
FldGetScrollPosition() *221*
FldGetScrollValues() *222*
FldGetSelection() *223*
FldGetTextAllocatedSize() *223*
FldGetTextHandle() *224*
FldGetTextHeight() *225*
FldGetTextLength() *225*
FldGetTextPtr() *226*
FldGetVisibleLines() *226*
FldGlueGetLineInfo() *1939*
FldGrabFocus 227, 455
FldGrabFocus() *227*
FldHandleEvent 50, 51
FldHandleEvent() *227*
fldHeightChangedEvent *51*, 230, 237, 313
FldInsert() *229*
FldMakeFullyVisible() *230*
FldNewField() *231*
FldPaste() *233*
FldRecalculateField() *233*
FldReleaseFocus() *234*
FldScrollable() *235*
FldScrollField() *235*
FldSendChangeNotification() *236*
FldSendHeightChangeNotification() *237*
FldSetAttributes() *237*
FldSetBounds() *238*
FldSetDirty() *239*
FldSetFont() *239*
FldSetInsertionPoint() *240*
FldSetInsPtPosition() *241*
FldSetMaxChars() *241*
FldSetMaxVisibleLines() *242*
FldSetScrollPosition() *242*
FldSetSelection() *243*
FldSetText() *244*
FldSetTextAllocatedSize() *246*
FldSetTextHandle() *246*
FldSetTextPtr() *248*
FldSetUsable() *249*
FldUndo() *250*
FldWordWrap() *250*
FloatMgr.h 757
FloatMgr.h (Palm OS 1.0) 2461
FloatMgrOld.h 2461
FlpAToF() *759*
FlpBase10Info() *760*
FlpBufferAToF() *761*
FlpBufferCorrectedAdd() *762*
FlpBufferCorrectedSub() *763*
FlpCompDouble *757*
FlpCorrectedAdd() *764*
FlpCorrectedSub() *765*
FlpDoubleBits *758*
FlpFToA() *766*
FlpGetExponent() *766*
FlpGetSign() *767*
FlpIsZero() *767*
FlpNegate() *767*
FlpSetNegative() *768*
FlpSetPositive() *768*
FlpVersion() *769*
flushing pen queue 1020
fntAppCustomBase *774*
FntAverageCharWidth 775, 777
FntAverageCharWidth() *785*
FntBaseLine 776, 778
FntBaseLine() *785*
FntCharHeight 775, 778
FntCharHeight() *785*
FntCharsInWidth() *786*
FntCharsWidth() *787*
FntCharWidth 772, 775, 778
FntCharWidth() *787*
FntDefineFont() *788*
FntDescenderHeight 776, 778
FntDescenderHeight() *789*
fntExtendedFormatMask 777, 779
FntGetFont 773
FntGetFont() *789*

FntGetFontPtr() *790*
FntGetScrollValues() *790*
FntGlueGetDefaultFontID 796
FntGlueGetDefaultFontID() *1940*
FntGlueWCharWidth 788, 793, 1930
FntGlueWidthToOffset 794, 1930
FntIsAppDefined() *791*
FntLineHeight 776, 778
FntLineHeight() *791*
FntLineWidth() *791*
fntMissingChar 779
FntSetFont 773
FntSetFont() *792*
fntTabChrWidth 779
FntWCharWidth 772, 775, 778, 787
FntWCharWidth() *793*
FntWidthToOffset 786
FntWidthToOffset() *793*
FntWordWrap() *794*
FntWordWrapReverseNLines() *795*
focus
    and modal window 1263
    FrmGetFocus 301
    FrmSetFocus 334
font resource (NFNT) 779
font resource, extended (nfnt) 782
Font.h 771
FontCharInfoPtr *771*, 771
FontCharInfoType *771*, 771, 775, 778
FontDefaultType 1940
FontDensityType *772*, 772, 778
FontID *773*, 773, 788
FontPtr *774*, 774
fonts
    and FldGetFont 219
FontSelect() *796*
FontSelect.h 771
FontType 774, *774*, 774, 780
FontTypeV2 772, *776*, 776, 782
Form Object 290
form objects
    FormType structure 274
    functions 289–344
form, active 299
Form.h 259
FormActiveStateType *260*, 330
FormAttrType *261*, 261
FormBitmapType *262*, 262
FormCheckResponseFuncPtr *345*
FormCheckResponseFuncType 288, 295
FormCheckResponseFuncType() *345*
FormEventHandlerPtr *346*
FormEventHandlerType 275
FormEventHandlerType() *346*
FormFrameType *263*, 263
FormGadgetAttrType *263*, 263
formGadgetDeleteCmd 296, 298, 346
formGadgetDrawCmd 342, 347
formGadgetEraseCmd 317, 347
formGadgetHandleEventCmd 347
FormGadgetHandlerType 52, 53, 296, 298, 317, 342
FormGadgetHandlerType() *346*
FormGadgetType 52, 53, *265*, 265, 346
FormGadgetTypeInCallback *267*
FormLabelType *268*, 268
FormLineType *269*, 269
formNavRscType *287*
FormObjAttrType *269*, 269
FormObjectKind *280*, 280
FormObjectType *270*, 270
FormObjListType *272*, 272
FormPopupType *272*, 272
FormPtr *273*, 273
FormRectangleType *273*, 273
FormTitleType *274*, 274
FormType *274*, 274, 1944
FplAdd 2461
FplAdd() *2461*
FplAToF 2462
FplAToF() *2462*
FplBase10Info 2462
FplBase10Info() *2462*
FplDiv 2463
FplDiv() *2463*
FplFloatToLong 2463
FplFloatToLong() *2463*
FplFloatToULong 2464

FplFloatToULong() *2464*
FplFree 2464
FplFree() *2464*
FplFToA 2464
FplFToA() *2464*
FplInit 2465
FplInit() *2465*
FplLongToFloat 2465
FplLongToFloat() *2465*
FplMul 2466
FplMul() *2466*
FplSub 2466
FplSub() *2466*
fprintf() *978*
fputc() *979*
fputs() *979*
frame type constants 1217
FrameBitsType *1215*, 1215
frames
    drawing in window 1240, 1243, 1269
FrameType *1216*, 1216
FrmActiveState *2373*
FrmAddSpaceForObject *2373*
FrmAlert 290
FrmAlert() *290*
frmBitmapObj *280*
FrmCloseAllForms 52
FrmCloseAllForms() *291*
frmCloseEvent *52*, 291, 312, 313
frmControlObj *280*
FrmCopyLabel() *291*
FrmCopyTitle() *292*
FrmCountObjectsInNavOrder() *293*
FrmCustomAlert() *293*
FrmCustomResponseAlert() *294*
FrmDeleteForm 346
FrmDeleteForm() *295*
frmDIAPolicyCustom *287*
frmDIAPolicyStayOpen *287*, 333
FrmDispatchEvent 312
FrmDispatchEvent() *296*
FrmDoDialog() *296*
FrmDrawForm 314, 347
FrmDrawForm() *297*
FrmEraseForm 58
FrmEraseForm() *298*
frmFieldObj *280*
frmFrameObj *280*
frmGadgetEnterEvent *52*, 313, 347
frmGadgetMiscEvent *53*, 314, 347
frmGadgetObj *281*
FrmGetActiveField() *298*
FrmGetActiveForm() *299*
FrmGetActiveFormID() *299*
FrmGetControlGroupSelection() *299*
FrmGetControlValue() *300*
FrmGetDIAPolicyAttr 287
FrmGetDIAPolicyAttr() *301*
FrmGetFirstForm() *301*
FrmGetFocus() *301*
FrmGetFormBounds() *302*
FrmGetFormId() *302*
FrmGetFormPtr() *302*
FrmGetGadgetData 266
FrmGetGadgetData() *303*
FrmGetLabel() *303*
FrmGetNavEntry() *304*
FrmGetNavOrder() *305*
FrmGetNavState() *306*
FrmGetNumberOfObjects() *306*
FrmGetObjectBounds 171, 173, 175
FrmGetObjectBounds() *307*
FrmGetObjectId() *307*
FrmGetObjectIndex() *308*
FrmGetObjectIndexFromPtr() *309*
FrmGetObjectPosition() *310*
FrmGetObjectPtr() *310*
FrmGetObjectType() *311*
FrmGetTitle() *311*
FrmGetUserModifiedState *2373*
FrmGetWindowHandle() *311*
FrmGlueGetActiveField() *1941*
FrmGlueGetDefaultButtonID() *1942*
FrmGlueGetHelpID() *1942*
FrmGlueGetLabelFont() *1942*
FrmGlueGetMenuBarID() *1943*

FrmGlueGetObjectUsable() *1944*
FrmGlueGetObjIDFromObjPtr() *1943*
FrmGlueNavDrawFocusRing() *1944*
FrmGlueNavGetFocusRingInfo() *1946*
FrmGlueNavIsSupported() *1947*
FrmGlueNavObjectTakeFocus() *1948*
FrmGlueNavRemoveFocusRing() *1949*
FrmGlueSetDefaultButtonID() *1949*
FrmGlueSetHelpID() *1950*
FrmGlueSetLabelFont() *1950*
frmGotoEvent *53*
FrmGotoForm 52, 54, 56
FrmGotoForm() *312*
frmGraffitiStateObj *281*
FrmGraffitiStateType *276*, 276
FrmHandleEvent 46, 52, 57, 70, 347
FrmHandleEvent() *312*
FrmHelp() *316*
FrmHideObject 178, 181, 264, 347
FrmHideObject() *316*
FrmInitForm() *317*
frmInvalidObjectId *287*, 308
frmLabelObj *280*
frmLineObj *280*
frmListObj *280*
frmLoadEvent *54*, 312, 333, 1292
FrmNavDrawFocusRing() *317*
frmNavFocusRingNoExtraInfo *287*
FrmNavFocusRingStyleEnum *281*
frmNavFocusRingStyleHorizontalBars *281*
frmNavFocusRingStyleInvalid *281*
frmNavFocusRingStyleObjectTypeDefault *281*
frmNavFocusRingStyleRound *281*
frmNavFocusRingStyleSquare *281*
FrmNavGetFocusRingInfo() *319*
FrmNavHeaderFlagsType *282*
FrmNavHeaderType *277*
FrmNavObjectFlagsType *284*
FrmNavObjectTakeFocus() *320*
FrmNavOrderEntryType *278*
FrmNavRemoveFocusRing() *321*
FrmNavStateFlagsType *285*
FrmNewBitmap() *322*
FrmNewForm() *323*
FrmNewGadget() *324*
FrmNewGsi() *325*
FrmNewLabel() *326*
frmNoSelectedControl *288*, 300
frmObjectFocusLostEvent *55*
frmObjectFocusTakeEvent *56*
frmOpenEvent 53, *56*, 275, 296, 297, 312, 327, 333, 1292
FrmPointInTitle() *327*
FrmPopupForm 54, 56
FrmPopupForm() *327*
frmPopupObj *281*
frmRectangleObj *280*
frmRedrawUpdateCode 58, *288*, 342
FrmRemoveObject() *328*
frmResponseCreate *288*, 345
frmResponseQuit *288*, 345
FrmRestoreActiveState() *329*
FrmReturnToForm() *329*
FrmSaveActiveState 329
FrmSaveActiveState() *330*
FrmSaveAllForms 57
FrmSaveAllForms() *330*
frmSaveEvent *57*, 330
frmScrollBarObj *281*
FrmSetActiveForm 75
FrmSetActiveForm() *331*
FrmSetCategoryLabel() *331*
FrmSetControlGroupSelection() *332*
FrmSetControlValue() *332*
FrmSetDIAPolicyAttr 287
FrmSetDIAPolicyAttr() *333*
FrmSetEventHandler() *334*
FrmSetFocus 455
FrmSetFocus() *334*
FrmSetGadgetData 266
FrmSetGadgetData() *335*
FrmSetGadgetHandler 266
FrmSetGadgetHandler() *336*
FrmSetMenu 395, 400
FrmSetMenu() *336*
FrmSetNavEntry() *337*

FrmSetNavOrder() *338*
FrmSetNavState() *339*
FrmSetNotUserModified *2373*
FrmSetObjectBounds 171, 173, 175
FrmSetObjectBounds() *340*
FrmSetObjectPosition() *340*
FrmSetTitle() *341*
FrmShowObject 191, 264, 347
FrmShowObject() *341*
frmTableObj *280*
frmTitleEnterEvent *57*, 57, 314
frmTitleObj *281*
frmTitleSelectEvent *57*, 314, 2406
frmUpdateEvent *58*, 288, 297, 314, 342
FrmUpdateForm 58, 59
FrmUpdateForm() *342*
FrmUpdateScrollers() *343*
FrmValidatePtr() *344*
FrmVisible() *344*
ftrErrNoSuchFeature 730, 731, 732, 734, 735
FtrGet 730
FtrGet() *730*
FtrGetByIndex 731
FtrGetByIndex() *731*
FtrPtrFree() *732*
FtrPtrNew() *732*
FtrPtrResize() *734*
FtrSet 735
FtrSet() *735*
FtrUnregister 735
FtrUnregister() *735*

G

gadget resource 264, 335
    extended 264, 336, 346
genericFormat *1221*
getchar() *980*
GetCharAttr() *600*
GetCharCaselessValue() *601*
GetCharSortValue() *602*
gethostname 1527
gets() *980*
GetSize 1589
GetSizeProcPtr() *1589*
GetSpace 1589
GetSpaceProcPtr() *1589*
global find 22, 25, 253–257, 1093, 1966, 1969
    FindDrawHeader 253
    FindGetLineBounds 254
    saving data 34
gotIt structure 123
goto (global find) 25
GoToParamsType 26
Graffiti 2 feature set 2436
Graffiti 2 recognizer
    enabling and disabling 1018
    EvtDequeuePenPoint 1017
Graffiti manager
    functions 797–807
Graffiti recognizer
    enabling and disabling 1018
    EvtDequeuePenPoint 1017
Graffiti Reference Dialog 428
Graffiti Shift
    functions 349–351
Graffiti.h 797
GraffitiReference.h 427
GraffitiShift.h 349
GraffitiUI.h 427
graphic characters 1075
GraphicControlType *172*, 172, 177
GraphicStatePtr 1223
grayPattern *1218*
grayUnderline *1219*
GrfAddMacro() *797*
GrfAddPoint() *798*
GrfCleanState() *798*
GrfDeleteMacro() *798*
GrfFilterPoints() *799*
GrfFindBranch() *799*
GrfFlushPoints() *800*
grfFtrInputAreaFlagActive 719
grfFtrInputAreaFlagCollapsible 719
grfFtrInputAreaFlagLiveInk 719
GrfGetAndExpandMacro() *800*
GrfGetGlyphMapping() *801*
GrfGetMacro() *801*

GrfGetMacroName() *802*
GrfGetNumPoints() *802*
GrfGetPoint() *803*
GrfGetState() *803*
GrfInitState() *804*
GrfMatch() *804*
GrfMatchGlyph() *805*
GrfProcessStroke() *806*
GrfSetState() *806*
groups of controls 299
GsiEnable() *349*
GsiEnabled() *349*
GsiInitialize() *350*
GsiSetLocation() *350*
GsiSetShiftState() *350*

H

hard reset 35
header line for global find 253
heap ID 845, 851
    of chunk 838
heaps
    compacting 843
    free bytes 844
    ROM based 844
height of text in field 225
Helper API 809
Helper.h 809
HelperNotifyEnumerateListType *809*, 809
HelperNotifyEventType 809, *810*, 810, 811, 813
HelperNotifyExecuteType *812*, 812, 814, 815
HelperNotifyValidateType *813*, 813
HelperServiceClass.h 809
HelperServiceEMailDetailsType 813, *814*, 814
HelperServiceSMSDetailsType 813, *815*
HelperServiceSMSDetailType 815
hidePrivateRecords *403*
HMSTime *196*
HostControl.h 1048
hostent 1460
HostGremlinIsRunning 1048
HotSync and sysAppLaunchCmdSyncNotify 34
HotSync operation 115
hwrMiscFlagExt115KIrOK 724
hwrMiscFlagExtHasExtLCD 724
hwrMiscFlagExtHasFlash 724
hwrMiscFlagExtHasFParms 724
hwrMiscFlagExtHasLiIon 724
hwrMiscFlagExtHasRailIO 724
hwrMiscFlagExtHasSWBright 724
hwrMiscFlagExtNeedsLpr 724
hwrMiscFlagExtSubIDMask 724
hwrMiscFlagHas3vRef 723
hwrMiscFlagHasAntennaSw 723
hwrMiscFlagHasBacklight 723
hwrMiscFlagHasBurrBrown 723
hwrMiscFlagHasCardIrDA 723
hwrMiscFlagHasCradleDetect 723
hwrMiscFlagHasJerryHW 723
hwrMiscFlagHasMbdIrDA 723
hwrMiscFlagHasMiscFlagExt 723
hwrMiscFlagHasSWContrast 723
hwrMiscFlagInvertLCDForBL 723
hwrMiscFlagNoRTCBug 723

I

icons 60
iconType 547
ID
    databases 629
    heap 845
IINetLibURLCrack() *1922*
iMessenger application 2396
IndexedColorType *1217*, 1217
INetCacheEntryType 1897
INetCacheInfoType 1895
inetCfgName... constants 1892–1893
inetCompressionType... constants 1882
inetCompressionTypeBitPacked *1882*
INetCompressionTypeEnum *1881*
inetCompressionTypeLZ77 *1882*
inetCompressionTypeNone *1882*
INetConfigNameType *1882*
inetContentType... constants 1883
inetContentTypeApplicationCML *1883*
INetContentTypeEnum *1883*

inetContentTypeImageGIF *1883*
inetContentTypeImageJPEG *1883*
inetContentTypeImagePalmOS *1883*
inetContentTypeOther *1883*
inetContentTypeTextHTML *1883*
inetContentTypeTextPlain *1883*
inetFtrNumVersion 714
inetHTTPAttr... constants 1884–1885
inetHTTPAttrCheckMailHi *1884*
inetHTTPAttrCheckMailLo *1884*
inetHTTPAttrContentCacheID *1885*
inetHTTPAttrContentLength *1885*
inetHTTPAttrContentLengthUncompressed *1885*
inetHTTPAttrContentLengthUntruncated *1885*
inetHTTPAttrContentVersion *1885*
INetHTTPAttrEnum *1883*
inetHTTPAttrErrDetail *1885*
inetHTTPAttrIncHTTP *1884*
inetHTTPAttrReason *1885*
inetHTTPAttrReqContentVersion *1884*
inetHTTPAttrReqSize *1885*
inetHTTPAttrResult *1884*
inetHTTPAttrRspSize *1884*
inetHTTPAttrWhichPart *1884*
INetLibCacheGetObject() *1894*
INetLibCacheList() *1896*
INetLibCheckAntennaState() *1898*
INetLibClose() *1899*
INetLibConfigAliasGet() *1899*
INetLibConfigAliasSet() *1900*
INetLibConfigDelete() *1901*
INetLibConfigIndexFromName() *1902*
INetLibConfigList() *1903*
INetLibConfigMakeActive() *1904*
INetLibConfigRename() *1904*
INetLibConfigSaveAs() *1905*
inetLibFtrCreator 714
INetLibGetEvent() *1906*
INetLibOpen() *1907*
INetLibSettingGet() *1908*
INetLibSettingSet() *1909*
INetLibSockClose() *1910*
INetLibSockConnect() *1911*
INetLibSockHTTPAttrGet() *1912*
INetLibSockHTTPAttrSet() *1913*
INetLibSockHTTPReqCreate() *1914*
INetLibSockHTTPReqSend() *1915*
INetLibSockOpen() *1916*
INetLibSockRead() *1918*
INetLibSockSettingGet() *1919*
INetLibSockSettingSet() *1920*
INetLibSockStatus() *1921*
INetLibURLCrack 1922
INetLibURLGetInfo() *1923*
INetLibURLOpen() *1924*
INetLibURLsAdd() *1926*
INetLibWiCmd() *1926*
INetMgr.h 41, 1881, 2315
inetOpenURLFlag... constants 1893–1894
inetScheme... constants 1886–1887
INetSchemeEnum *1885*
inetSchemeFile *1886*
inetSchemeFTP *1886*
inetSchemeGopher *1886*
inetSchemeHTTP *1886*
inetSchemeHTTPS *1886*
inetSchemeMac *1887*
inetSchemeMail *1887*
inetSchemeMailTo *1886*
inetSchemeNews *1886*
inetSchemePalm *1886*
inetSchemePalmCall *1886*
inetSetting... constants 1887–1888
inetSettingBaseStationID *1888*
inetSettingCacheRef *1887*
inetSettingCacheSize *1887*
inetSettingContentVersion *1888*
inetSettingContentWidth *1888*
inetSettingConvAlgorithm *1888*
INetSettingEnum *1887*
inetSettingMaxRspSize *1888*
inetSettingNetLibConfig *1887*
inetSettingNoPersonalInfo *1888*
inetSettingRadioID *1887*
inetSettingUserName *1888*
inetSockReadyEvent *59*

inetSockSetting... constants 1889–1890
inetSockSettingCompressionType *1889*
inetSockSettingCompressionTypeID *1889*
inetSockSettingContentType *1889*
inetSockSettingContentTypeID *1889*
inetSockSettingContentVersion *1890*
inetSockSettingData *1889*
inetSockSettingDataHandle *1889*
inetSockSettingDataOffset *1890*
INetSockSettingEnum *1888*
inetSockSettingFlags *1890*
inetSockSettingIndexURL *1890*
inetSockSettingReadTimeout *1890*
inetSockSettingScheme *1889*
inetSockSettingSockContext *1889*
inetSockSettingTitle *1890*
inetSockSettingURL *1890*
inetSockStatusChangeEvent *59*
inetStatus... constants 1891
inetStatusAcquiringNetwork *1891*
inetStatusClosed *1891*
inetStatusClosingConnection *1891*
inetStatusConnected *1891*
inetStatusConnecting *1891*
INetStatusEnum *1890*
inetStatusNameResolved *1891*
inetStatusNew *1891*
inetStatusPrvInvalid *1891*
inetStatusReceivingResponse *1891*
inetStatusResolvingName *1891*
inetStatusResponseReceived *1891*
inetStatusSendingRequest *1891*
inetStatusWaitingForResponse *1891*
InetURLInfo type 1924
inetURLInfoFlag... constants 1893
InetURLType 1922
informationAlert *279*
initialization 30
input area state 881, 884, 885, 886
Insertion Point 353
insertion point functions 353–356
insertion points
  and FldGetInsPtPosition 220
  and FldGrabFocus 227
  and FldReleaseFocus 234
  and FldSetInsertionPoint 240
  displayed in field 217
insertion sort 1049
InsPoint.h 353
InsPtEnable() *353*
InsPtEnabled() *354*
InsPtGetHeight() *354*
InsPtGetLocation() *354*
InsPtSetHeight() *355*
InsPtSetLocation() *355*
interaction mode 285
international manager 2394
Internet library 1881
intlErrInvalidSelector 1315
IntlGetRoutineAddress 1315
IntlGetRoutineAddress() *1314*
IntlGlue.h 1951
IntlGlueGetRoutineAddress() *1951*
IntlMgr.h 1307, 1314, 1315, 1951
intlMgrBestFit 729
intlMgrExists 728
intlMgrStrict 729
IntlSetRoutineAddress() *1315*
inverting characters in draw window 1260
inverting line in draw window 1260
IR Library 2388
IR manager 1423
IrAdvanceCredit() *1434*
IrBind() *1435*
IrCallbackParms *1427*
IrClose() *1436*
IrConnect *1423*
IrConnectIrLap() *1436*
IrConnectReq() *1437*
IrConnectRsp() *1438*
IrDataReq() *1439*
IrDisconnectIrLap() *1440*
IrDiscoverReq() *1441*
irFtrCreator 714
irFtrNumVersion 714
irGetScanningMode 1429
irGetStatistics 1429
IrIAS_Add() *1449*

IrIAS_GetInteger() *1450*
IrIAS_GetIntLsap() *1450*
IrIAS_GetObjectID() *1451*
IrIAS_GetOctetString() *1451*
IrIAS_GetOctetStringLen() *1451*
IrIAS_GetType() *1452*
IrIAS_GetUserString() *1452*
IrIAS_GetUserStringCharSet() *1452*
IrIAS_GetUserStringLen() *1453*
IrIAS_Next() *1453*
IrIAS_Query() *1454*
IrIAS_SetDeviceName() *1455*
IrIAS_StartResult() *1456*
IrIASObject *1425*
IrIasQuery *1426*
IrIasQueryCallback 1456
IrIasQueryCallBack() *1456*
IrIsIrLapConnected() *1441*
IrIsMediaBusy() *1442*
IrIsNoProgress() *1442*
IrIsRemoteBusy() *1442*
irlib.h 1423
IrLocalBusy() *1443*
IrMaxRxSize() *1443*
IrMaxTxSize() *1444*
IrOpen() *1445*
IrPacket *1424*
irRestoreScanning 1429
irSetBaudMask 1430
IrSetConTypeLMP() *1445*
IrSetConTypeTTP() *1446*
IrSetDeviceInfo() *1446*
irSetScanningMode 1430
irSetSerialMode 1431
irSetSupported 1431
IrStatsType *1428*
irSuppressScanning 1431
IrTestReq() *1447*
IrUnbind() *1448*
ISO 639 827
iterate structure 124

J

Japanese feature set 2396

K

kAdnEnableDebugIndicator *517*
kAdnEnableFullDebugging *517*
kAdnEnableMasterSwitch *517*
kAdnEnableShowSafeFatalAlerts *517*
kAdnErrUnsupportedCall *519*
kAdnSemihostArmReportException *517*
kAdnSemihostArmWrite0 *518*
kAdnSemihostDebugEnableGet *518*
kAdnSemihostDebugEnableGetSupported *518*
kAdnSemihostDebugEnableSet *518*
kAdnSemihostLicenseeSpecific *518*
kAdnSemihostNativeRegister *518*
kAdnSemihostNativeUnregister *518*
kAdnSemihostPUDConnect *518*
kAdnSemihostPUDDebugBreak *518*
kAdnSemihostPUDDownloadEvent *518*
kAdnSemihostPUDFaultNotification *518*
kAdnSemihostPUDModulePostLoad *518*
kAdnSemihostPUDModulePostUnload *518*
kAdnSemihostPUDModuleTableUpdate *518*
kAdnSemihostPUDProcessCreate *518*
kAdnSemihostPUDProcessDestroy *518*
kAdnSemihostPUDProfilerEvent *518*
kAdnSemihostPUDThreadCreate *518*
kAdnSemihostPUDThreadDestroy *518*
kAttnCommandCustomEffect 120
kAttnCommandDrawDetail 120
kAttnCommandDrawList 120
kAttnCommandGoThere 120
kAttnCommandGotIt 120
kAttnCommandIterate 120
kAttnCommandPlaySound 120
kAttnCommandSnooze 120
kAttnFlagsAllBits 125
kAttnFlagsAlwaysCustomEffect 125
kAttnFlagsAlwaysLED 125
kAttnFlagsAlwaysSound 125
kAttnFlagsAlwaysVibrate 125
kAttnFlagsCapabilitiesMask 130

kAttnFlagsCustomEffectBit 125
kAttnFlagsEverything 125
kAttnFlagsHasCustomEffect 131
kAttnFlagsHasLED 130
kAttnFlagsHasSound 130
kAttnFlagsHasVibrate 130
kAttnFlagsLEDBit 125
kAttnFlagsNoCustomEffect 126
kAttnFlagsNoLED 126
kAttnFlagsNoSound 126
kAttnFlagsNothing 126
kAttnFlagsNoVibrate 126
kAttnFlagsSoundBit 125
kAttnFlagsUserSettingsMask 130
kAttnFlagsUserWantsCustomEffect 131, 907
kAttnFlagsUserWantsLED 131, 907
kAttnFlagsUserWantsSound 131, 907
kAttnFlagsUserWantsVibrate 131, 907
kAttnFlagsUseUserSettings 125
kAttnFlagsVibrateBit 125
kAttnFtrCapabilities 130, 718
kAttnFtrCreator 130, 718
kAttnLevelInsistent 128
kAttnLevelSubtle 128
kBitmapScalingOff 1298
kCncDeviceKindGPRS *1335*
kCncDeviceKindGSM *1335*
kCncDeviceKindLocalNetwork *1336*
kCncDeviceKindModem *1336*
kCncDeviceKindPhone *1336*
kCncDeviceKindSerial *1336*
kCncErrDBAccessFailed 1348, 1349, 1350, 1351, 1352, 1353, 1354
kCncErrProfileParamNotFound 1348, 1349, 1350, 1351, 1354
kCncFtrCncMgrCreator 714
kCncFtrCncMgrVersion 714, 2401
kCncMgrVersion 2401
kCncNotifyCreateRequest 85
kCncNotifyDeleteRequest 85
kCncNotifyModifyRequest 85
kCncNotifyUpdateListRequest 85
kCncParam_GPRS_APNName *1331*
kCncParam_GPRS_DataCompression *1331*
kCncParam_GPRS_HeaderCompression *1331*
kCncParam_GPRS_QOS_Delay *1331*
kCncParam_GPRS_QOS_Mean *1331*
kCncParam_GPRS_QOS_Peak *1331*
kCncParam_GPRS_QOS_Precedence *1332*
kCncParam_GPRS_QOS_Reliability *1332*
kCncParam_GSM_BearerServices *1330*
kCncParam_GSM_ConnectionElement *1330*
kCncParam_GSM_Speed *1330*
kCncParam_PSDCreator *1330*, 1333
kCncParam_PSDName *1330*, 1333
kCncParam_PSDParameterBuffer *1330*
kCncParam_PSDType *1330*, 1333
kCncParamBaud 1324, 1332
kCncParamBluetoothDeviceAddr 1324, 1332
kCncParamBluetoothDeviceName 1324, 1332
kCncParamBuffer *1335*
kCncParamCountryIndex 1324, 1325, 1332
kCncParamDeviceKind 1332
kCncParamDialingMode 1332
kCncParamFlowControl 1332
kCncParamInitString 1332
kCncParamIntlModemCountryStringList 1326
kCncParamIntlModemResetStringList 1326, 1328
kCncParamInvisible 1327, 1333
kCncParamLocked 1327, 1333
kCncParamName 1327, 1333
kCncParamNoDetails 1327, 1333
kCncParamNonEditable 1328, 1333
kCncParamPort 1328, 1333
kCncParamReadOnly 1328, 1333
kCncParamReceiveTimeOut *1329*, 1333
kCncParamResetString 1328, 1333
kCncParamSerialPortFlags 1328
kCncParamString *1335*
kCncParamSystemFlag *1335*, 1338
kCncParamSystemFlags 1329, 1333, 1335
kCncParamThirdPartiesRange 1338
kCncParamTimeOut 1329, 1333
kCncParamTTCreator *1330*, 1333
kCncParamTTType *1330*, 1333
kCncParamUInt16 *1335*
kCncParamUInt32 *1335*

kCncParamUInt8 *1335*
kCncParamVersion 1329, 1333
kCncParamVolume 1329, 1333
kCncProfileClassicResetStringSize 1333
kCncProfileNameSize 1333
kCncProfileUsualInitStringSize 1332
kCncProfileUsualResetStringSize 1333
kCncProfileVersion 1329
kCoordinatesDouble 1227
kCoordinatesNative 1227
kCoordinatesOneAndAHalf 1227
kCoordinatesQuadruple 1227
kCoordinatesStandard 1227
kCoordinatesTriple 1227
kDensityDouble *544*, 772, 776
kDensityLow *544*, 772, 776
kDensityOneAndAHalf *544*, 772
kDensityQuadruple *544*
kDensityTriple *544*
kDrvrVersion 1579
kDrvrVersion3 1579
kDrvrVersion4 1579
key manager functions 819–820
key queue
  size 1026
keyBitPageDown 819
keyBitPageUp 819
keyBitPower 819
keyboard display 1050
KeyCurrentState() *819*
keyDownEvent *60*, 114, 227, 275, 314, 379, 391, 397, 398, 1019, 2392
keyHoldEvent *62*
KeyMgr.h 819
KeyRates() *820*
KeySetMask() *820*
keyUpEvent *63*
kFrmNavHeaderFlagsAppFocusStartState *282*
kFrmNavHeaderFlagsAppFocusStartStateValue *282*
kFrmNavHeaderFlagsAutoGenerated *282*
kFrmNavHeaderFlagsDefaultStartStateValue *282*
kFrmNavHeaderFlagsInvalidStartStateValue *282*
kFrmNavHeaderFlagsObjectFocusStartState *283*
kFrmNavHeaderFlagsObjectFocusStartStateValue *283*
kFrmNavHeaderFlagsStartStateMask *283*
kFrmNavInfoVersion *288*
kFrmNavObjectFlagsForceInteractionMode *284*
kFrmNavObjectFlagsIsBigButton *284*
kFrmNavObjectFlagsSkip *284*
kFrmNavStateFlagsInteractionMode *285*
kFrmNavStateFlagsObjectFocusMode *285*
kHelperNotifyActionCodeEnumerate 809, 811
kHelperNotifyActionCodeExecute 811, 812
kHelperNotifyActionCodeValidate 811, 813
kHelperServiceClassIDEMail 813, 814, 816
kHelperServiceClassIDFax 813, 816
kHelperServiceClassIDSMS 813, 815, 816
kHelperServiceClassIDVoiceDial 813, 816
kMaxCountryNameLen 828, 830
kMaxCurrencyNameLen 828, 830
kMaxCurrencySymbolLen 828, 830, 832
kPalmOSEmulatorFeatureCreator 726
kPalmOSEmulatorFeatureNumber 726
kSmsFtrNumVersion 714
kSslClose *2218*
kSslConsume *2218*
kSslContextCreate *2218*
kSslContextDestroy *2218*
kSslContextGetLong *2218*
kSslContextGetPtr *2218*
kSslContextSetLong *2218*
kSslContextSetPtr *2218*
kSslDBName *2219*
kSslFlush *2218*
kSslLibClose *2218*
kSslLibCreate *2218*
kSslLibCreator *2219*
kSslLibDestroy *2218*
kSslLibGetLong *2218*
kSslLibGetPtr *2218*
kSslLibName *2218*
kSslLibOpen *2218*
kSslLibSetLong *2218*
kSslLibSetPtr *2218*
kSslLibSleep *2218*

kSslLibType *2219*
kSslLibWake *2218*
kSslOpen *2218*
kSslPeek *2218*
kSslRead *2218*
kSslReceive *2218*
kSslSend *2219*
kSslWrite *2219*
kTelCallConnected *1670*
kTelCallConnecting *1670*
kTelCallDisconnecting *1670*
kTelCallIdle *1670*
kTelCallIncoming *1670*
kTelCallIncomingAck *1670*
kTelCallNotificationPriority *1677*
kTelCallRedial *1670*
kTelCallServiceData *1671*
kTelCallServiceVoice *1671*
kTelCallTypeIncoming *1670*
kTelCallTypeOutgoing *1670*
kTelCfgServiceId *1676*
kTelDataCallClass *1677*
kTelDtcServiceId *1676*
kTelEmcLaunchCmdCallBusy *1681*
kTelEmcLaunchCmdCallConnect *1681*
kTelEmcLaunchCmdCallError *1681*
kTelEmcLaunchCmdCallReleased *1681*
kTelEmcServiceId *1676*
kTelFaxCallClass *1677*
kTelInfiniteDelay *1679*
kTelInfPhoneBrand *1675*
kTelInfPhoneModel *1675*
kTelInfPhoneRevision *1675*
kTelInfServiceId *1677*
kTelInvalidAppId *1680*
kTelLastLaunchCode *1681*
kTelLocationSeparator *1680*
kTelMaxPhoneNumberLen *1680*
kTelMgrDatabaseCreator *1678*
kTelMgrDatabaseType *1678*
kTelMgrLibName *1678*
kTelMgrVersion *1678*
kTelMgrVersionBuild *1678*
kTelMgrVersionFix *1678*
kTelMgrVersionMajor *1678*
kTelMgrVersionMinor *1678*
kTelNwkAutomaticSearch *1744*
kTelNwkCDMA *1744*
kTelNwkGSM *1744*
kTelNwkManualSearch *1744*
kTelNwkPDC *1744*
kTelNwkServiceId *1676*
kTelNwkTDMA *1744*
kTelOemServiceId *1677*
kTelPhbAdaptorPhonebook *1862*
kTelPhbFirstOemPhonebook *1862*
kTelPhbFixedPhonebook *1862*
kTelPhbLastDialedPhonebook *1862*
kTelPhbPhonePhonebook *1862*
kTelPhbServiceId *1677*
kTelPhbSimAndPhonePhonebook *1862*
kTelPhbSimPhonebook *1862*
kTelPowBatteryFault *1669*
kTelPowBatteryNotPowered *1669*
kTelPowBatteryPowered *1669*
kTelPowNoBattery *1669*
kTelPowServiceId *1676*
kTelSmsAPIVersion *1678*, 1816, 1821
kTelSmsCMTMessageType *1826*
kTelSmsCPTMessageType *1827*
kTelSmsDefaultProtocol *1824*
kTelSmsDSRMessageForwarded *1827*
kTelSmsDSRMessageReplaced *1827*
kTelSmsDSRPermBadDestination *1828*
kTelSmsDSRPermDeleteByAdm *1828*
kTelSmsDSRPermDeletedByOrigSME *1828*
kTelSmsDSRPermInternetworkError *1828*
kTelSmsDSRPermOther *1828*
kTelSmsDSRPermRPError *1827*
kTelSmsDSRPermServiceUnavailable *1828*
kTelSmsDSRPermSMNotExist *1828*
kTelSmsDSRPermUnobtainable *1828*
kTelSmsDSRPermValidityExpired *1828*
kTelSmsDSRSuccess *1827*
kTelSmsDSRTempCongestion *1827*
kTelSmsDSRTempOther *1827*

kTelSmsDSRTempServiceRejected *1827*
kTelSmsDSRTempServiceUnavailable *1827*
kTelSmsDSRTempSMEBusy *1827*
kTelSmsDSRTempSMEError *1827*
kTelSmsEmailProtocol *1824*
kTelSmsErmesProtocol *1824*
kTelSmsFaxProtocol *1824*
kTelSmsLaunchCmdIncomingMessage *1680*
kTelSmsManualAckDeliveryType *1828*
kTelSmsMultiPartExtensionTypeId *1823*
kTelSmsNbs2ExtensionTypeId *1823*
kTelSmsNbsExtensionTypeId *1823*
kTelSmsNotificationPriority *1677*
kTelSmsPagingProtocol *1824*
kTelSmsPrivacyConfidential *1826*
kTelSmsPrivacyNotRestricted *1826*
kTelSmsPrivacyRestricted *1826*
kTelSmsPrivacySecret *1826*
kTelSmsServiceId *1676*
kTelSmsStatusReportDeliveryType *1828*
kTelSmsStorageAdaptor *1825*
kTelSmsStorageFirstOem *1825*
kTelSmsStoragePhone *1825*
kTelSmsStorageSIM *1825*
kTelSmsUrgencyEmergency *1826*
kTelSmsUrgencyNormal *1826*
kTelSmsUrgencyUrgent *1826*
kTelSmsVMNMessageType *1827*
kTelSmsVoiceProtocol *1824*
kTelSmsX400Protocol *1824*
kTelSndServiceId *1677*
kTelSpcCallingLineId *1680*
kTelSpcLaunchCmdCallBusy *1680*
kTelSpcLaunchCmdCallConnect *1680*
kTelSpcLaunchCmdCallerIDAvailable *1680*
kTelSpcLaunchCmdCallError *1681*
kTelSpcLaunchCmdCallReleased *1680*
kTelSpcLaunchCmdIncomingCall *1680*
kTelSpcServiceId *1676*
kTelSpeechCallClass *1677*
kTelStyPin1CodeId *1730*
kTelStyPin2CodeId *1730*
kTelStyPuk1CodeId *1730*
kTelStyPuk2CodeId *1730*
kTelStyReady *1730*
kTelStyServiceId *1676*
kTelTelephonyEvent *1679*
kTelTelephonyNotification *85*, *1679*
kTextPaddingOff 1298
kTextScalingOff 1298
kTxtConvertStateSize 1063

L

labelTableItem 435
LanguageType *823*, 823, 869
largeBoldFont *774*, 796, 1941, 2386
largeFont *773*, 796, 1941
launch codes
    summary 3, 79, 809
    SysBroadcastActionCode 1037
launch flags 37
Launcher.h 427
ledFont *774*
LEVENT_DATA_IND *1432*
LEVENT_DISCOVERY_CNF *1432*
LEVENT_LAP_CON_CNF *1432*
LEVENT_LAP_CON_IND *1432*
LEVENT_LAP_DISCON_IND *1432*
LEVENT_LM_CON_CNF *1432*
LEVENT_LM_CON_IND *1433*
LEVENT_LM_DISCON_IND *1433*
LEVENT_PACKET_HANDLED *1433*
LEVENT_STATUS_IND *1433*
LEVENT_TEST_CNF *1433*
LEVENT_TEST_IND *1434*
libEvtHookKeyMask 61
libPalmOSGlue.a 1929
lightGrayPattern *1218*
LineInfoPtr *212*, 212
LineInfoType *212*, 212
lines
    erasing 1246
    inverting 1260
list objects
    and pen tracking 69
    creating category list 147
    drawItemsCallback 361, 371

functions 361–371
structure 358
List.h 357
ListAttrType *357*, 357
ListDrawDataFuncType() *372*
ListPtr 361
lists
setting items 1043
ListType *358*
ListType structure 358
lLanguageName constants 827
lmAnyCountry 834
lmAnyLanguage 203, 834, 1139
lmChoice... constants 830
lmErrBadLocaleIndex 829
lmErrBadLocaleSettingChoice 829
lmErrSettingDataOverflow 829
lmErrUnknownLocale 833
LmGetLocaleSetting 833
LmGetLocaleSetting() *828*
LmGetNumLocales() *832*
LmGlueGetLocaleSetting 832, 1930
LmGlueGetNumLocales 833, 1930
LmGlueGetSystemLocale() *1952*
LmGlueLocaleToIndex 834, 1930
LmLocaleToIndex() *833*
LmLocaleType *824*, 824, 830, 833, 907
local ID 846, 854
from chunk handle 841
locale 907
LocaleMgr.h 823
Localize.h 823, 1307
LocGetNumberSeparators 16, 824
LocGetNumberSeparators() *1316*
locking chunk 838
locking system 35
longDateStrLength 1131
lookup 31
example 31
LstDrawList() *361*
lstEnterEvent *64*, 64, 65, 314, 364
LstEraseList() *362*
lstExitEvent *64*, 65
LstGetNumberOfItems() *362*
LstGetSelection() *363*
LstGetSelectionText() *363*
LstGetTopItem() *364*
LstGetVisibleItems() *364*
LstGlueGetDrawFunction() *1952*
LstGlueGetFont() *1953*
LstGlueGetItemsText() *1953*
LstGlueGetTopItem 1930
LstGlueSetFont() *1954*
LstGlueSetIncrementalSearch() *1954*
LstHandleEvent 64, 65
LstHandleEvent() *364*
LstMakeItemVisible() *365*
LstNewList() *366*
LstPopupList() *367*
LstScrollList() *367*
lstSelectEvent 64, *65*
LstSetDrawFunction() *368*
LstSetHeight() *368*
LstSetListChoices() *369*
LstSetPosition() *370*
LstSetSelection() *370*
LstSetTopItem() *371*

M

maskPrivateRecords *403*
maxFieldLines 230
maxFieldTextLen 241
maxStrIToALen 1000
MdmDial 1457
MdmDial() *1457*
mdmErrBusy 1457
mdmErrCmdError 1457
mdmErrNoDCD 1457
mdmErrNoTone 1457
mdmErrUserCan 1457
MdmHangUp 1458
MdmHangUp() *1458*
MeasurementSystemType 830, *897*
MemCardInfo() *835*
MemCmp() *836*
MemDebugMode() *837*
memErrChunkLocked 840

memErrInvalidParam 732, 734, 839, 842, 850
memErrNotEnoughSpace 165, 732, 734, 735, 839, 1032, 1038, 1512, 1640
MemGluePtrNew() *1955*
MemHandleCardNo() *837*
MemHandleDataStorage 837
MemHandleDataStorage() *837*
MemHandleFree() *838*
MemHandleHeapID() *838*
MemHandleLock 838
MemHandleLock() *838*
MemHandleNew 839
MemHandleNew() *839*
MemHandleResize 839
MemHandleResize() *839*
MemHandleSetOwner 840
MemHandleSetOwner() *840*
MemHandleSize 841
MemHandleSize() *841*
MemHandleSsetOwner 840
MemHandleToLocalID 841
MemHandleToLocalID() *841*
MemHandleUnlock 842
MemHandleUnlock() *842*
MemHeapCheck 842
MemHeapCheck() *842*
MemHeapCompact() *842*
MemHeapDynamic 843
MemHeapDynamic() *843*
memHeapFlagReadOnly 844
MemHeapFlags 843
MemHeapFlags() *843*
MemHeapFreeBytes 844
MemHeapFreeBytes() *844*
MemHeapID 845
MemHeapID() *845*
MemHeapScramble 845
MemHeapScramble() *845*
MemHeapSize 846
MemHeapSize() *846*
MemLocalIDKind 846
MemLocalIDKind() *846*
MemLocalIDToGlobal() *846*
MemLocalIDToGlobalNear 846
MemLocalIDToLockedPtr 847
MemLocalIDToLockedPtr() *847*
MemLocalIDToPtr 847
MemLocalIDToPtr() *847*
MemMove 848
MemMove() *848*
MemNumCards 848
MemNumCards() *848*
MemNumHeaps 845, 849
MemNumHeaps() *849*
MemNumRAMHeaps 849
MemNumRAMHeaps() *849*
memory
    and FldCompactText 214
    and FldFreeMemory 218
    and FldSetText 245, 248
memory blocks, comparing 836
memory card information 835
memory manager
    debugging mode 837, 855
MemoryMgr.h 835
MemPtrCardNo 849
MemPtrCardNo() *849*
MemPtrDataStorage 850
MemPtrDataStorage() *850*
MemPtrFree 850
MemPtrFree() *850*
MemPtrHeapID() *851*
MemPtrNew 851
MemPtrNew() *851*
MemPtrRecoverHandle 852
MemPtrRecoverHandle() *852*
MemPtrResize 852
MemPtrResize() *852*
MemPtrSetOwner 853
MemPtrSetOwner() *853*
MemPtrSize 853
MemPtrSize() *853*
MemPtrSsetOwner 853
MemPtrToLocalID 854
MemPtrToLocalID() *854*
MemPtrUnlock 854
MemPtrUnlock() *854*

MemSet 854
MemSet() *854*
MemSetDebugMode 855
MemSetDebugMode() *855*
MemStoreInfo 856
MemStoreInfo() *856*
menu objects
  See also menus 382
  fields 383
  structure 382
Menu.h 377
MenuAddItem 384, 392
MenuAddItem() *387*
MenuBarAttrType *377*, 377
MenuBarPtr *382*, 382
MenuBarType *382*, 382, 388
menuButtonCause 68, 399
menuChr 388, 397, 398, 399, 401
menuCloseEvent *65*
MenuCmdBarAddButton 66, 108, 379, 389
MenuCmdBarAddButton() *389*
MenuCmdBarButtonType *378*, 378, 380
MenuCmdBarDisplay() *392*
MenuCmdBarGetButtonData() *393*
menuCmdBarMaxTextLength 393
menuCmdBarOnLeft 389
menuCmdBarOnRight 389
menuCmdBarOpenEvent *66*, 109, 228, 314, 390, 392, 393, 398
menuCmdBarResultChar *380*
menuCmdBarResultMenuItem *380*, 391
menuCmdBarResultNone *380*
menuCmdBarResultNotify *380*
MenuCmdBarResultType *380*, 380
MenuCmdBarType 378, *380*, 380, 384
menuCommandCause 68, 399
MenuDispose() *394*
MenuDrawMenu 378
MenuDrawMenu() *394*
MenuEraseStatus 380, 393
MenuEraseStatus() *395*
menuErrNoMenu 388
menuErrNotFound 388
menuErrOutOfMemory 389
menuErrSameId 388
menuErrTooManyItems 390
menuEvent 58, *66*, 314, 379, 391, 398
MenuGetActiveMenu() *397*
MenuHandleEvent 66, 67, 108, 378, 391, 394
MenuHandleEvent() *397*
MenuHideItem() *399*
MenuInit() *400*
MenuItemType *384*, 384, 388
menuOpenEvent *67*, 388, 399, 401
MenuPullDownPtr *385*, 385
MenuPullDownType 384, *385*, 385
menus
  FrmSetMenu 336
  functions 387–400
MenuSeparatorChar 385, 387, 388
MenuSetActiveMenu 400
MenuSetActiveMenu() *400*
MenuSetActiveMenuRscID() *401*
MenuShowItem() *401*
missing character symbol 787
modal window 367, 1233, 1263
modem 1457
ModemMgr.h 1457
modified field objects 216
multi-byte characters 1067, 1071, 1094, 1095, 1096
  attributes 1081
  comparison 1068, 1082
  converting 1106
  encodings support 1061–1109
  searching 1093, 1966, 1969
  size 1080
multiple preferences 917

N

narrowTextTableItem 437, 443, 444, 460
nativeFormat *1221*
navigation mode 285
NavSelectorFrmCountObjectsInNavOrder *288*
NavSelectorFrmGetNavEntry *288*
NavSelectorFrmGetNavOrder *288*
NavSelectorFrmGetNavState *288*
NavSelectorFrmNavDrawFocusRing *288*
NavSelectorFrmNavGetFocusRingInfo *288*

NavSelectorFrmNavObjectTakeFocus *288*
NavSelectorFrmNavRemoveFocusRing *288*
NavSelectorFrmSetNavEntry *288*
NavSelectorFrmSetNavOrder *289*
NavSelectorFrmSetNavState *289*
net library
    closing 1472, 1485
    open count 1513
    opening 1510, 1511
netCfgNameCTPWireless *1467*
netCfgNameCTPWireline *1467*
netCfgNameDefault *1467*
netCfgNameDefWireless *1467*
netCfgNameDefWireline *1467*
netConfigIndexCurSettings 1513
NetConfigNamePtr 1459
NetConfigNameType *1459*, 1459
netErrAlreadyOpen 1510, 1512
netErrAlreadyOpenWithAnotherConfig 1512
netErrAuthFailure 1506
netErrBadScript 1505
netErrBufTooSmall 1479, 1483, 1498, 1512, 1526
netErrBufWrongSize 1498, 1504, 1526, 1529
netErrConfigAliasErr 1480, 1512
netErrConfigBadName 1481, 1482
netErrConfigCantDelete 1480, 1481, 1512
netErrConfigCantPointToAlias 1475
netErrConfigEmpty 1480, 1512
netErrConfigNotAlias 1474, 1475
netErrConfigNotFound 1477, 1480, 1512
netErrConfigTooMany 1482
netErrDNSAborted 1487, 1489, 1491
netErrDNSAllocationFailure 1487, 1488, 1491
netErrDNSBadName 1487, 1488, 1491
netErrDNSBadProtocol 1487, 1489, 1491
netErrDNSFormat 1487, 1489, 1491
netErrDNSImpossible 1487, 1489, 1491
netErrDNSIrrelevant 1487, 1489, 1491
netErrDNSLabelTooLong 1487, 1488, 1491
netErrDNSNameTooLong 1487, 1488, 1490
netErrDNSNIY 1487, 1489, 1491
netErrDNSNonexistantName 1487, 1489, 1491
netErrDNSNoPort 1487, 1489, 1491
netErrDNSNoRecursion 1487, 1489, 1491
netErrDNSNoRRS 1487, 1489, 1491
netErrDNSNotInLocalCache 1487, 1489, 1491
netErrDNSRefused 1487, 1489, 1491
netErrDNSServerFailure 1487, 1489, 1491
netErrDNSTimeout 1487, 1488, 1491
netErrDNSTruncated 1487, 1489, 1491
netErrDNSUnreachable 1487, 1488, 1491
netErrInterfaceDown 1522, 1524
netErrInterfaceNotFound 1493, 1494, 1495, 1498, 1504, 1505, 1512, 1522, 1524
netErrInternal 1537, 1538
netErrInvalidInterface 1496
netErrInvalidSettingSize 1529, 1542, 1544
netErrIPCantFragment 1521, 1524
netErrIPktOverflow 1522, 1524
netErrIPNoDst 1522, 1524
netErrIPNoRoute 1522, 1524
netErrIPNoSrc 1522, 1524
netErrMessageTooBig 1521, 1524
netErrNoInterfaces 1483, 1510, 1512, 1536, 1538
netErrNoMoreSockets 1540
netErrNoMultiPacketAddr 1548
netErrNoMultiPktAddr 1522, 1524
netErrNotOpen 1473, 1485, 1487, 1488, 1490, 1492, 1495, 1505, 1507, 1515, 1516, 1519, 1521, 1524, 1531, 1533, 1534, 1535, 1536, 1538, 1540, 1542, 1544, 1548, 1549
netErrOutOfCmdBlocks 1474, 1475, 1477, 1478, 1480, 1481, 1482, 1483, 1512, 1522, 1524, 1534, 1535, 1537, 1538, 1540, 1548
netErrOutOfMemory 1485, 1510, 1512, 1540
netErrOutOfPackets 1522, 1524
netErrOutOfResources 1538
netErrParamErr 1474, 1475, 1477, 1480, 1481, 1485, 1507, 1512, 1515, 1516, 1521, 1524, 1531, 1533, 1534, 1535, 1536, 1538, 1540, 1542, 1544, 1548
netErrPortInUse 1537, 1538
netErrPPPAddressRefused 1506
netErrPPPTimeout 1506
netErrPrefNotFound 1480, 1496, 1497, 1504, 1510, 1512, 1526
netErrQuietTimeNotElapsed 1537, 1538
netErrReadOnlySetting 1504, 1529
netErrSocketAlreadyConnected 1534, 1537, 1538

netErrSocketBusy 1536, 1538
netErrSocketClosedByRemote 1521, 1524, 1531, 1533, 1534, 1537, 1538
netErrSocketInputShutdown 1548
netErrSocketNotConnected 1521, 1524, 1531
netErrSocketNotListening 1531
netErrSocketNotOpen 1485, 1515, 1516, 1521, 1524, 1531, 1533, 1534, 1535, 1536, 1538, 1542, 1544, 1548
netErrStillOpen 1473
netErrTimeout 1485, 1487, 1488, 1490, 1492, 1512, 1515, 1516, 1521, 1524, 1531, 1532, 1534, 1535, 1536, 1538, 1539, 1542, 1544, 1548
netErrTooManyInterfaces 1493
netErrTooManyTCPConnections 1537
netErrUnimplemented 1498, 1504, 1507, 1531, 1542, 1544
netErrUnknownProtocol 1492
netErrUnknownService 1493
netErrUnknownSetting 1497, 1504, 1526, 1529
netErrUnreachableDest 1522, 1524
netErrUserCancel 1485, 1505, 1515
netErrWouldBlock 1485, 1515, 1516, 1522, 1524, 1537
netErrWrongSocketType 1531, 1537, 1538, 1542, 1544
netFDIsSet 1519
netFDSet 1519
netFDSetSize 1520
NetFDSetType 1518
netFDZero 1519
netFtrCommandBlocks 715
netFtrCreator 715
netFtrNumVersion 715
NetHostInfoBufType *1460*, 1460
NetHostInfoType *1460*, 1460
NetHToNL 1470
NetHToNL() *1470*
NetHToNS() *1471*
netIFMediaDown 110
netIFMediaUp 109
NetIFSettingEnum 1497, 1498, 1504
netIOFlagDontRoute *1468*
netIOFlagOutOfBand *1468*
netIOFlagPeek *1468*
NetIOParamType 1517
NetIOVecPtr 1517
NetIOVecType 1517
NetIPAddr 1460, 1471, 1472
NetLibAddrAToIN 1471
NetLibAddrAToIN() *1471*
NetLibAddrINToA 1472
NetLibAddrINToA() *1472*
NetLibClose 1472
NetLibClose() *1472*
NetLibConfigAliasGet 1474
NetLibConfigAliasGet() *1474*
NetLibConfigAliasSet 1475
NetLibConfigAliasSet() *1475*
NetLibConfigDelete 1476
NetLibConfigDelete() *1476*
NetLibConfigIndexFromName 1477
NetLibConfigIndexFromName() *1477*
NetLibConfigList 1478
NetLibConfigList() *1478*
NetLibConfigMakeActive 1479
NetLibConfigMakeActive() *1479*
NetLibConfigRename 1480
NetLibConfigRename() *1480*
NetLibConfigSaveAs 1481
NetLibConfigSaveAs() *1481*
NetLibConnectionRefresh() *1483*
NetLibDmReceive 1484
NetLibDmReceive() *1484*
NetLibFinishCloseWait 1485
NetLibFinishCloseWait() *1485*
NetLibGetHostByAddr 1486
NetLibGetHostByAddr() *1486*
NetLibGetHostByName 1488
NetLibGetHostByName() *1488*
NetLibGetMailExchangeByName 1490
NetLibGetMailExchangeByName() *1490*
NetLibGetServByName 1492
NetLibGetServByName() *1492*
NetLibIFAttach 1493
NetLibIFAttach() *1493*
NetLibIFDetach 1494

NetLibIFDetach() *1494*
NetLibIFDown 1495
NetLibIFDown() *1495*
NetLibIFGet 1496
NetLibIFGet() *1496*
NetLibIFSettingGet 1497
NetLibIFSettingGet() *1497*
NetLibIFSettingSet 1504
NetLibIFSettingSet() *1504*
NetLibIFUp 1505
NetLibIFUp() *1505*
NetLibMaster 1506
NetLibMaster() *1506*
NetLibOpen 1510
NetLibOpen() *1510*
NetLibOpenConfig 1511
NetLibOpenConfig() *1511*
NetLibOpenCount 1513
NetLibOpenCount() *1513*
NetLibReceive 1514, 1552
NetLibReceive() *1514*
NetLibReceivePB 1516
NetLibReceivePB() *1516*
NetLibSelect 1518
NetLibSelect() *1518*
NetLibSend 1520, 1553
NetLibSend() *1520*
NetLibSendPB 1523
NetLibSendPB() *1523*
NetLibSettingGet 1525
NetLibSettingGet() *1525*
NetLibSettingSet 1529
NetLibSettingSet() *1529*
NetLibSocketAccept 1530, 1531
NetLibSocketAccept() *1530*
NetLibSocketAddr 1532
NetLibSocketAddr() *1532*
NetLibSocketBind 1533
NetLibSocketBind() *1533*
NetLibSocketClose 1535
NetLibSocketClose() *1535*
NetLibSocketConnect 1536, 1552
NetLibSocketConnect() *1536*
NetLibSocketListen 1537, 1538
NetLibSocketListen() *1537*
NetLibSocketOpen 1539, 1552
NetLibSocketOpen() *1539*
NetLibSocketOptionGet 1541
NetLibSocketOptionGet() *1541*
NetLibSocketOptionSet 1543
NetLibSocketOptionSet() *1543*
NetLibSocketShutdown 1547
NetLibSocketShutdown() *1547*
NetLibTracePrintF 1548
NetLibTracePrintF() *1548*
NetLibTracePutS 1549
NetLibTracePutS() *1549*
NetMasterEnum 1506
netMasterICMPStats command 1509
netMasterInterfaceInfo command 1507
netMasterInterfaceStats command 1508
netMasterIPStats command 1509
NetMasterPBPtr 1506
netMasterTCPStats command 1509
netMasterTraceEventGet command 1509
netMasterUDPStats command 1509
NetMgr.h 1459
NetNToHL 1550
NetNToHL() *1550*
NetNToHS 1550
NetNToHS() *1550*
NetServInfoBufType *1461*, 1461
NetServInfoType *1462*, 1462
NetSettingEnum 1525, 1526, 1529
NetSocket.c 1551
NetSocketAddrEnum *1462*, 1462
netSocketAddrINET 1540
NetSocketAddrINType *1463*, 1463
netSocketAddrRaw 1540
NetSocketAddrRawType *1463*, 1463
NetSocketAddrType *1464*, 1464
netSocketDirBoth 1547
NetSocketDirEnum 1547
netSocketDirInput 1547
netSocketDirOutput 1547
NetSocketLingerType 1545

netSocketNoticeCallback *1466*
netSocketNoticeConnectInbound *1469*
netSocketNoticeConnectOutbound *1469*
netSocketNoticeErr *1468*
netSocketNoticeEvent *1466*
netSocketNoticeMailbox *1466*
netSocketNoticeNotify *1466*
netSocketNoticeTCPClosed *1469*
netSocketNoticeTCPReceive *1468*
netSocketNoticeTCPRemoteClosed *1468*
netSocketNoticeTCPTransmit *1468*
NetSocketNoticeType *1464*, 1464
netSocketNoticeUDPReceive *1468*, 1468
netSocketNoticeWake *1466*
NetSocketOptEnum 1541, 1543, 1545
NetSocketOptLevelEnum 1541, 1543, 1545
netSocketProtoIPRAW 1539
netSocketProtoIPTCP 1539
netSocketProtoIPUDP 1539
NetSocketRef *1465*, 1539
netSocketTypeDatagram *1466*
NetSocketTypeEnum *1465*, 1465
netSocketTypeRaw *1466*
netSocketTypeStream *1466*
netTracingAppMsgs *1469*
netTracingAppPktIP 1469
netTracingData *1470*
netTracingData40 *1470*
netTracingErrors *1469*
netTracingFuncs *1469*
netTracingIFHi *1470*
netTracingIFLow *1470*
netTracingIFMid *1470*
netTracingMsgs 1468, *1469*
netTracingPktIP *1469*
netTracingPkts *1469*
NetUReadN() *1551*
NetUTCPOpen() *1552*
NetUWriteN() *1553*
never *898*
new serial manager 2398, 2399
nfApostropheComma *825*
nfApostrophePeriod *825*
nfCommaPeriod *825*
NFNT resource 779
nfnt resource 782
nfPeriodComma *825*
nfSpaceComma *825*
nilEvent *68*, 68, 1029, 1030
noButtonFrame *167*
noFocus 275, *288*, 301, 334
noListSelection 363
noMenuItemSelection 383, 387
noMenuSelection 383, 387
noPreferenceFound 911, 913
noteTextTableItem 444
NoticeTypeEnum *1466*, 1466
Notification *1680*
notification manager 2408
NotifyMgr.h 79, 859
noUnderline *1219*
NumberFormatType *824*, 824, 831
numericGraffitiSilkscreenArea 1024
numericTableItem 435
numLockMask 61
numUneditableCategories 148, 150, 157

O

Object focus mode 285
off-screen windows 1231
olume Constants 957
omErrBadOverlayDBName 876
omErrBaseRequiresOverlay 611
omErrDatabaseRequiresOverlay 654
omErrInvalidLocaleIndex 872
omErrNoNextSystemLocale 873
omErrUnknownLocale 611, 875, 876
omFtrCreator 726, 727, 870
omFtrDefaultLocale 727, 870
omFtrShowErrorsFlag 726, 870
OmGetCurrentLocale() *871*
OmGetIndexedLocale() *872*
OmGetNextSystemLocale() *873*
OmGetRoutineAddress() *874*
OmGetSystemLocale 833
OmGetSystemLocale() *874*

OmGlueGetCurrentLocale 871, 1930
OmGlueGetSystemLocale 875, 1930
OmLocaleToOverlayDBName() *875*
OmLocaleType 824, *869*, 869
OmOverlayDBNameToLocale() *876*
omOverlayDBType 654
omOverlayRscID 870
omOverlayRscType 870
OmSelector 874
OmSetSystemLocale() *877*
open count of net library 1513
opening net library 1510, 1511
optionKeyMask 61
OverlayMgr.h 869, 874
overlays 654

P

Palm OS 2.0 feature set 2382, 2459
Palm OS 3.0 feature set 2385, 2438, 2440
Palm OS 3.1 feature set 2389
Palm OS 3.2 feature set 2392
Palm OS 3.5 feature set 2402, 2421, 2423
Palm OS 4.0 feature set 2409
Palm OS 5.0 feature set 2426
Palm OS 5.1 feature set 2431
Palm OS 5.3SC feature set 2437
Palm OS 5.4 feature set 2438
PalmLocale.h 823, 825, 827
PalmOSGlue.lib 1929, 2390, 2394
panel list (SysCreatePanelList) 1040
password functions 879
Password.h 879
PatternType *1218*, 1218
PceNativeCall() *1316*
PdiDefineReaderDictionary() *2331*
PdiDefineResizing() *2332*
PdiDefineWriterDictionary() *2332*
PdiDictionary *2315*
PdiEnterObject() *2334*
PdiLibClose() *2334*
PdiLibOpen() *2335*
PdiParameterPairTest() *2335*
PdiReaderDelete() *2336*
PdiReaderNew() *2337*
PdiReaderType *2316*
PdiReadParameter() *2338*
PdiReadProperty() *2338*
PdiReadPropertyField() *2339*
PdiReadPropertyName() *2341*
PdiSetCharset() *2342*
PdiSetEncoding() *2343*
PdiWriteBeginObject() *2343*
PdiWriteEndObject() *2344*
PdiWriteParameter() *2345*
PdiWriteParameterStr() *2347*
PdiWriteProperty() *2348*
PdiWritePropertyBinaryValue() *2348*
PdiWritePropertyFields() *2349*
PdiWritePropertyStr() *2350*
PdiWritePropertyValue() *2351*
PdiWriterDelete() *2352*
PdiWriterNew() *2353*
PdiWriterType *2318*
pen
    current status 1021
Pen Input Manager 881
pen manager functions 895–896
pen queue
    flushing 1020
    size 1026
PenBtnInfoType 1022
PenCalibrate() *895*
penDownEvent 46, 50, 52, 57, 64, *68*, 73, 180, 227, 315, 364, 397, 422
PenInputMgr.h 881
PenMgr.h 895
penMoveEvent *69*
PenResetCalibration() *896*
penUpEvent *69*, 1018
PhoneLookup.h 497
PhoneNumberLookup 501
PhoneNumberLookup() *501*
PhoneNumberLookupCustom() *502*
PilotMain() *1060*
pinAPIVersion1_0 2435
pinAPIVersion1_1 2435
pinErrInvalidParam 333, 886, 1291

pinErrNoSoftInputArea 886, 1291
pinFtrAPIVersion 2434
PINGetInputAreaState 881
PINGetInputAreaState() *884*
PINGetInputTriggerState 882
PINGetInputTriggerState() *884*
pinInputAreaClosed 881
pinInputAreaNone 882
pinInputAreaOpen 881
pinInputAreaUser 882
pinInputTriggerDisabled 882
pinInputTriggerEnabled 882
pinInputTriggerNone 882
PINSetInputAreaState 75, 100, 881, 882
PINSetInputAreaState() *885*
PINSetInputTriggerState 882
PINSetInputTriggerState() *886*
pixelFormat565 *545*
pixelFormat565LE *545*
pixelFormatIndexed *545*
pixelFormatIndexedLE *545*
PixelFormatType *544*
PluginCallbackProcType *1555*, 1555
PluginCmdPtr *1556*, 1556
PluginCmdType *1556*, 1556
PluginExecCmdType *1556*, 1556, 1559
PluginInfoPtr *1557*, 1557
PluginInfoType *1557*, 1557, 1558
pluginMaxNumOfCmds 1558
pluginNetLibCallUIProc 1560, 1562, 1564
pluginNetLibCheckCancelStatus 1559, 1563
pluginNetLibConnLog 1559, 1564
pluginNetLibDoNothing 1559, 1562
pluginNetLibGetSerLibRefNum 1560, 1564
pluginNetLibGetUserName 1559, 1563
pluginNetLibGetUserPwd 1559, 1563
pluginNetLibPromptUser 1559, 1563
pluginNetLibReadBytes 1559, 1562
pluginNetLibWriteBytes 1559, 1563
PointType *921*
popSelectEvent *70*, 313, 315
popup list 367
popupTriggerCtl *170*
popupTriggerTableItem 436
port... constants 1579
poweredOnKeyMask 61
prefAlarmSoundVolume 946, 947
preferenceDataVer10 911
preferenceDataVer11 911
preferenceDataVer12 912
preferenceDataVer2 911
preferenceDataVer3 911
preferenceDataVer4 911
preferenceDataVer5 911
preferenceDataVer6 911
preferenceDataVer8 911
preferenceDataVer9 911
preferenceDataVerLatest 912, 2391
preferenceDataVersion 2391
preferences
    changing with launch codes 33
    multiple application preferences 917
Preferences.h 823, 824, 897
prefGameSoundVolume 946, 947
PrefGetAppPreferences 912, 2384
PrefGetAppPreferences() *912*
PrefGetAppPreferencesV10 914
PrefGetAppPreferencesV10() *914*
PrefGetPreference 108, 915
PrefGetPreference() *915*
PrefGetPreferences 915
PrefGetPreferences() *915*
PrefOpenPreferenceDB 916
PrefOpenPreferenceDB() *916*
PrefOpenPreferenceDBV10 916
PrefOpenPreferenceDBV10() *916*
PrefSetAppPreferences 917
PrefSetAppPreferences() *917*
PrefSetAppPreferencesV10 918
PrefSetAppPreferencesV10() *918*
PrefSetPreference 919
PrefSetPreference() *919*
PrefSetPreferences 919
PrefSetPreferences() *919*
prefShowPrivateRecords 404, 405
prefSysSoundVolume 946, 947
prefTimeZone 203

PrgCallbackData 412, 413
PrgCallbackFunc() *412*
PrgHandleEvent() *407*
PrgStartDialog() *408*
PrgStartDialogV31() *409*
PrgStopDialog() *410*
PrgUpdateDialog() *411*
PrgUserCancel() *412*
printf() *980*
PrivateRecords.h 403
privateRecordViewEnum *403*, 403, 905
procedure alarms 512
progress manager 2386
Progress Manager callback function 412
Progress.h 407
pushButtonCtl *170*
putc() *981*
putchar() *981*
puts() *982*
PwdExists 879
PwdExists() *879*
PwdRemove 879
PwdRemove() *879*
PwdSet 880
PwdSet() *880*
PwdVerify 880
PwdVerify() *880*

Q

query callback function 1456

R

RAM-based heaps 849
RctCopyRectangle() *922*
RctGetIntersection() *922*
RctInsetRectangle() *923*
RctOffsetRectangle() *924*
RctPtInRectangle() *925*
RctSetRectangle() *925*
read callback function 1406
Read68KUnaligned32() *568*
ReadProc 1371
records
    deleting 626
    detaching 627
    ID 629
    retrieving information 663
Rect.h 921
rectangleButtonFrame *167*
RectanglePtr 923
rectangles
    copying 922
    erasing 1247
    intersecting 922
    moving 924
    resizing 923
    scrolling 1287
RectangleType *921*, 923
reinitializing dynamic memory heap 1055
repeat control object
    and ctlRepeatEvent 47
repeating button 47
repeatingButtonCtl *170*
reset 35, 1055
ResGlueLoadConstant 504, 1930
ResLoadConstant() *503*
ResLoadForm() *504*
ResLoadMenu 505
ResLoadMenu() *505*
resource database (SysCurAppDatabase) 1041
resource ID 603
resource type 604, 631
resources
    retrieving 641
    retrieving information 669
    searching for 670
resumeSleepChr 114, 115
RGBColorType *545*, 545
ROM-based heaps 844, 849
ROM-based records 659, 661

S

Sampled Sound Application-Defined Functions 971
Sampled Sound Functions 958
Sampled Sound Structures, Constants, and Data Types 950

SclDrawScrollBar() *421*
sclEnterEvent *71*, 315, 422
sclExitEvent *71*, 423
SclGetScrollBar() *421*
SclHandleEvent 71, 72
SclHandleEvent() *422*
sclRepeatEvent *72*, 315, 423
    and sclExitEvent 71
SclSetScrollBar() *423*
scptLaunchCmdDoNothing *1558*
scptLaunchCmdExecuteCmd 3, 1556, *1559*
scptLaunchCmdListCmds 3, 1557, *1558*
ScrDisplayMode 1286, 2388, 2405
ScrDisplayModeOperation 1286
ScreenAttrType 1281
screenFormat *1221*
ScriptPlugin.h 1555
ScriptPluginLaunchCodesEnum *1558*, 1558
ScriptPluginSelectorProc() *1561*
ScriptPluginSelectorProcPtr 1561
scroll arrows
    FrmUpdateScrollers 343
scroll position in field 221
scrollbar functions 421–425
scrollbar objects
    fields 419
    in tables 457
    structure 419
ScrollBar.h 417
ScrollBarAttrType *417*, 417
ScrollBarPtr *418*, 418
ScrollBarRegionType 418
ScrollBarType *419*, 419
scrolling rectangle in window 1287
ScrOperation 1226
searching for string 256
secret records, removing *667*
SecSelectViewStatus() *404*
Security application 35
SecurityAutoLockType *897*, 897
SecVerifyPW() *404*
SelDay.h 195
SelectDay 2384
SelectDay() *198*
selectDayByDay 198
selectDayByMonth 198
selectDayByWeek 198
SelectDayType *195*
SelectDayV10() *199*
selection in field 223
SelectOneTime() *199*
selectorTriggerCtl *170*
SelectTime 201
SelectTime() *200*
SelectTimeV33() *201*
SelectTimeZone() *202*
SelTime.h 195
SelTimeZone.h 195
separatorItemSelection 387
SerClearErr 1635, 1638
SerClearErr() *1635*
SerClose() *1636*
SerControl() *1637*
serCtlBreakStatus (in SerCtlEnum) 1634
serCtlEmuSetBlockingHook (in SerCtlEnum) 1634
SerCtlEnum *1633*, 1633
serCtlFirstReserved (in SerCtlEnum) 1633
serCtlHandshakeThreshold (in SerCtlEnum) 1634
serCtlLAST (in SerCtlEnum) 1634
serCtlMaxBaud (in SerCtlEnum) 1634
serCtlStartBreak (in SerCtlEnum) 1633
serCtlStartLocalLoopback (in SerCtlEnum) 1634
serCtlStopBreak (in SerCtlEnum) 1634
serCtlStopLocalLoopback (in SerCtlEnum) 1634
serDev... constants 1601
serErrAlreadyOpen 1636, 1640
serErrBadParam 1637, 1640, 1649
serErrBadPort 1626
serErrConfigurationFailed 1620, 1626
serErrLineErr 1635, 1636, 1639, 1641, 1642, 1643, 1644
serErrNotOpen 1626, 1627, 1628, 1636, 1637, 1638, 1649
serErrNotSupported 1626, 1627, 1628
serErrStillOpen 1636
serErrTimeOut 1626, 1641, 1642, 1644, 1645, 1646, 1647
serFncConsole 1573, 1598

serFncDebugger 1573, 1597
serFncHotSync 1573, 1598
serFncPPPSession 1573, 1597
serFncSLIPSession 1573, 1597
serFncTelephony 1573, 1598
serFncUndefined 1573, 1597
SerGetSettings() *1638*
SerGetStatus() *1638*
serial capabilities constants 1601
Serial Library 1640
serial port feature constants 1579
serial settings constants 1601
serial status constants 1604
SerialDrvr.h 1567
SerialLinkMgr.h 1651
SerialMgr.h 1593
SerialMgrOld.h 1633
SerialVdrv.h 1567
serLineError... constants 1639
SerOpen 1639
SerOpen() *1639*
serPortConsolePort 1599
serPortCradlePort 1599
serPortCradleRS232Port 1599
serPortCradleUSBPort 1599
serPortIrPort 1599
serPortLocalHotSync 1599
SerReceive() *1641*
SerReceive10() *1642*
SerReceiveCheck 1643
SerReceiveCheck() *1643*
SerReceiveFlush 1643
SerReceiveFlush() *1643*
SerReceiveWait() *1644*
SerSend 1645
SerSend() *1645*
SerSend10() *1646*
SerSendFlush() *1647*
SerSendWait 1647
SerSendWait() *1647*
SerSetReceiveBuffer 1648
SerSetReceiveBuffer() *1648*
SerSetSettings 1648
SerSetSettings() *1648*
SerSettingsType *1635*, 1635
servent 1462
sethostname 1527
SetR10ToDataPointer() *526*
shiftKeyMask 61
showPrivateRecords *403*
SignalCheckPtr() *1590*
silk-screen buttons
    EvtGetPenBtnList 1022
SilkscreenAreaType 1024
silkscreenRectGraffiti 1024
silkscreenRectScreen 1024
SioAddCommand() *982*
SioClearScreen() *990*
SioExecCommand() *991*
Siofgetc 977, 980
Siofgetc() *983*
Siofgets 978
Siofgets() *983*
Siofprintf 978
Siofprintf() *984*
Siofputc 979, 981
Siofputc() *984*
Siofputs 979
Siofputs() *985*
SioFree() *991*
Siogets 980
Siogets() *985*
SioHandleEvent() *992*
SioInit() *992*
SioMain() *993*
Sioprintf 980
Sioprintf() *986*
Sioputs 982
Sioputs() *986*
Siosystem 988
Siosystem() *987*
Siovfprintf 989
Siovfprintf() *987*
SleepEventParamType 114
SliderControlType *174*, 174
sliderCtl *170*

SlkClose 1651
SlkClose() *1651*
SlkCloseSocket 1652
SlkCloseSocket() *1652*
slkErrAlreadyOpen 1653
slkErrBadParam 1657
slkErrBuffer 1655
slkErrChecksum 1655
slkErrNotOpen 1651
slkErrOutOfSockets 1654
slkErrSocketNotOpen 1652, 1653, 1655, 1656, 1657, 1658
slkErrTimeOut 1655, 1656
slkErrWrongDestSocket 1655
SlkFlushSocket 1652
SlkFlushSocket() *1652*
SlkOpen 1653
SlkOpen() *1653*
SlkOpenSocket 1653
SlkOpenSocket() *1653*
SlkPktHeaderType 1657
SlkReceivePacket 1654
SlkReceivePacket() *1654*
SlkSendPacket 1656
SlkSendPacket() *1656*
SlkSetSocketListener 1656
SlkSetSocketListener() *1656*
SlkSocketListenType 1657
SlkSocketPortID() *1658*
SlkSocketSetTimeout 1658
SlkSocketSetTimeout() *1658*
SlkWriteDataType 1656
slOff *898*
slOn *898*
SmsLib.h 2171
SmsParamsType *2171*, 2171
SmsPrefType *2173*, 2173
SmsReceiveCDMAParamsType *2174*, 2174
SmsReceiveGSMParamsType *2176*, 2176
SmsReceiveParamsType *2176*, 2176
SmsReceiveTDMAParamsType *2178*, 2178
SMSReportParamsType 2178
SmsReportParamsType *2178*
SmsSendCDMAParamsType *2179*, 2179
SmsSendGSMParamsType *2180*
SMSSendParamsType *2180*, 2180
SmsSendTDMAParamsType *2182*, 2182
sndAlarm *936*
sndAlarmVolume *958*
SndBlockingFuncType() *949*
SndCallbackInfoType *927*
sndClick *936*
sndCmdFreqDurationAmp *934*, 939
sndCmdFrqOn *934*, 939
SndCmdIDTag *933*
SndCmdIDType *929*, 933
sndCmdNoteOn *934*, 940
sndCmdQuiet *934*, 940
SndCommandType *928*
SndComplFuncType() *950*
sndConfirmation *936*
SndCreateMidiList() *937*
SndDoCmd() *937*
sndError *936*
sndFlagAsync 957
sndFlagNormal 957
sndFlagSync 956
sndFloat *954*
sndFloatBig *954*
sndFloatLittle *954*
sndFloatOpposite *954*
sndFormatAAC *953*
sndFormatDVI_ADPCM *953*
sndFormatIMA_ADPCM *953*
sndFormatMP3 *953*
sndFormatOGG *953*
sndFormatPCM *953*
SndFormatTag 952
SndFormatType *950*
sndFtrIDVersion 729, 952, 2424
sndGameVolume *958*
SndGetDefaultVolume() *940*
sndInfo *936*
sndInput *956*
sndInt16 *955*
sndInt16Big *955*

sndInt16Little *955*
sndInt16Opposite *955*
sndInt32 *955*
sndInt32Big *955*
sndInt32Little *955*
sndInt32Opposite *955*
sndInt8 *955*
SndInterruptSmfIrregardless() *941*
sndMaxAmp 903
sndMgrVersionNum 952, 2424
SndMidiListItemType *929*
sndMidiNameLength *934*
SndMidiRecHdrType *930*
sndMidiRecSignature *935*
SndMidiRecType *930*
sndMono *956*
sndOutput *956*
sndPanCenter 957
sndPanFullLeft 957
sndPanFullRight 957
SndPlayResource() *958*
SndPlaySMF 941
SndPlaySmf *935*
SndPlaySmf() *941*
SndPlaySmfIrregardless() *944*
SndPlaySmfResource() *946*
SndPlaySmfResourceIrregardless() *947*
SndPlaySystemSound() *948*
SndPtr 950, *951*, 952
SndSampleTag *953*
SndSampleType *951*
SndSetDefaultVolume() *948*
SndSmfCallbacksType *931*
SndSmfChanRangeType *931*
sndSmfCmdDuration *935*
SndSmfCmdEnum *932*, 935
sndSmfCmdPlay *935*
SndSmfOptionsType *932*
sndSmfPlayAllMilliSec *935*
sndStartUp *936*
sndStereo *956*
SndStreamBufferCallback() *971*
SndStreamCreate() *960*
SndStreamCreateExtended() *964*
SndStreamDelete() *966*
SndStreamDeviceControl 2376
SndStreamGetPan() *967*
SndStreamGetVolume() *967*
SndStreamMode *951*
SndStreamModeTag *955*
SndStreamPause() *968*
SndStreamRef *951*
SndStreamSetPan() *968*
SndStreamSetVolume() *969*
SndStreamStart() *970*
SndStreamStop() *971*
SndStreamVariableBufferCallback() *973*
SndStreamWidth *952*
SndStreamWidthTag *956*
SndSysBeepType *933*, 935
sndSystemVolume *958*
sndUInt8 *955*
sndWarning *936*
sockaddr 1464
sockaddr_in 1463
socket 1552
socket listener 1655
socket listener procedure 1655, 1657
Socket Notification *86*, 86
soft constant 503
soft reset 35, 1055
solidUnderline *1219*
sorting array elements 1049
sorting text 602
SortRecordInfoType *604*, 604
Sound *952*
Sound Manager 927
sound manager 2387
Sound Resource Playback Flags 956
SoundLevelTypeV20 *898*
SoundMgr.h 927
sprintf (StrPrintF) 1008
sprintf() *988*
SrmClearErr() *1605*
SrmClose() *1605*
SrmControl 1594, 1600, 1601, 1611

SrmControl() *1606*
srmControlCustom 1609
srmCtlCustom *1597*
srmCtlEmuSetBlockingHook *1596*
SrmCtlEnum *1594*, 1594, 1606
srmCtlGetBaudRate *1595*
srmCtlGetCtsTimeout *1595*
srmCtlGetDTRAsserted *1596*
srmCtlGetFlags *1595*
srmCtlGetOptimalTransmitSize *1596*
srmCtlIrDADisable *1596*
srmCtlIrDAEnable *1595*
srmCtlRxDisable *1596*
srmCtlRxEnable *1596*
srmCtlSetBaudRate *1595*
srmCtlSetCtsTimeout *1595*
srmCtlSetDTRAsserted *1596*
srmCtlSetFlags *1595*
srmCtlSetYieldPortCallback *1596*
srmCtlSetYieldPortRefCon *1596*
srmCtlStartBreak *1595*
srmCtlStartLocalLoopback *1595*
srmCtlStopBreak *1595*
srmCtlStopLocalLoopback *1595*
srmCtlSystemReserved *1597*
srmCtlUserDef *1596*
SrmCustomControl() *1609*
srmDefaultCTSTimeout 1611
SrmExtOpen 1598, 1612, 1616
SrmExtOpen() *1610*
SrmExtOpenBackground 1598, 1618
SrmExtOpenBackground() *1612*
SrmGetDeviceCount() *1613*
SrmGetDeviceInfo 1593
SrmGetDeviceInfo() *1614*
SrmGetStatus 1604, 1620, 1621, 1623
SrmGetStatus() *1615*
SrmOpen 1598, 1612
SrmOpen() *1616*
SrmOpenBackground 1598
SrmOpenBackground() *1617*
SrmOpenConfigType *1597*, 1597, 1611, 1612
SrmPrimeWakeupHandler 1630, 1631
SrmPrimeWakeupHandler() *1618*
SrmRcvQType *1570*, 1570
SrmReceive 1623, 1625, 1631
SrmReceive() *1619*
SrmReceiveCheck 1623
SrmReceiveCheck() *1620*
SrmReceiveFlush() *1621*
SrmReceiveWait 1625
SrmReceiveWait() *1622*
SrmReceiveWindowClose 1631
SrmReceiveWindowClose() *1623*
SrmReceiveWindowOpen 1631
SrmReceiveWindowOpen() *1624*
SrmSend 1628
SrmSend() *1625*
SrmSendCheck() *1626*
SrmSendFlush() *1627*
SrmSendWait() *1628*
SrmSetReceiveBuffer 1611, 1622
SrmSetReceiveBuffer() *1629*
srmSettings... constants 1601
SrmSetWakeupHandler 1618, 1631
SrmSetWakeupHandler() *1630*
srmStatus... constants 1604
sslAlertAccessDenied *2212*
sslAlertBadCertificate *2212*
sslAlertBadRecordMac *2212*
sslAlertCertificateExpired *2212*
sslAlertCertificateRevoked *2212*
sslAlertCertificateUnknown *2212*
sslAlertCloseNotify *2212*
sslAlertDecodeError *2212*
sslAlertDecompressionFailure *2212*
sslAlertDecryptError *2212*
sslAlertDecryptionFailed *2212*
sslAlertExportRestricion *2212*
sslAlertHandshakeFailure *2212*
sslAlertIllegalParameter *2212*
sslAlertInsufficientSecurity *2212*
sslAlertInternalError *2213*
sslAlertNoCertificate *2213*
sslAlertNoRenegotiation *2213*
sslAlertProtocolVersion *2213*

sslAlertRecordOverflow *2213*
sslAlertUnexpectedMessage *2213*
sslAlertUnknownCa *2213*
sslAlertUnsupportedCertificate *2213*
sslAlertUserCancled *2213*
sslArgInfoAlert *2207*
sslArgInfoCert *2207*
sslArgInfoHandshake *2207*
sslArgInfoReadAfter *2208*
sslArgInfoReadBefore *2208*
sslArgInfoWriteAfter *2208*
sslArgInfoWriteBefore *2208*
sslAttrAppInt32 *2252*
sslAttrAppPtr *2252*
sslAttrAutoFlush *2252*
sslAttrBufferedReuse *2252*
sslAttrCertPeerCert *2252*
sslAttrCertPeerCommonName *2252*
sslAttrCertSslVerify *2252*
sslAttrClientCertRequest *2252*
sslAttrCompat *2252*
sslAttrCspCipherSuite *2252*
sslAttrCspCipherSuiteInfo *2252*
sslAttrCspCipherSuites *2252*
sslAttrCspSslSession *2252*
sslAttrDontSendShutdown *2252*
sslAttrDontWaitForShutdown *2252*
sslAttrError *2252*
sslAttrErrorState *2252*
sslAttrHsState *2252*
SslAttribute *2189*
sslAttrInfoCallback *2252*
sslAttrInfoInterest *2252*
sslAttrIoFlags *2252*
sslAttrIoSocket *2252*
sslAttrIoStruct *2252*
sslAttrIoTimeout *2252*
sslAttrLastAlert *2252*
sslAttrLastApi *2252*
sslAttrLastIo *2252*
sslAttrLibAppInt32 *2253*
sslAttrLibAppPtr *2253*
sslAttrLibAutoFlush *2253*
sslAttrLibBufferedReuse *2253*
sslAttrLibCompat *2253*
sslAttrLibDontSendShutdown *2253*
sslAttrLibDontWaitForShutdown *2253*
sslAttrLibInfoCallback *2253*
sslAttrLibInfoInterest *2253*
sslAttrLibMode *2253*
sslAttrLibProtocolVersion *2253*
sslAttrLibRbufSize *2253*
sslAttrLibReadStreaming *2253*
sslAttrLibVerifyCallback *2253*
sslAttrLibWbufSize *2253*
sslAttrMode *2253*
sslAttrProtocolVersion *2253*
sslAttrRbufSize *2253*
sslAttrReadBufPending *2253*
sslAttrReadOutstanding *2253*
sslAttrReadRecPending *2253*
sslAttrReadStreaming *2253*
sslAttrSessionReused *2253*
sslAttrStreaming *2253*
sslAttrVerifyCallback *2253*
sslAttrWbufSize *2253*
sslAttrWriteBufPending *2253*
SslCallback *2190*
SslCallbackFunc() *2241*
SslCipherSuiteInfo *2191*
SslClose() *2219*
sslCloseDontSendShutdown *2202*
sslCloseDontWaitForShutdown *2203*
sslCloseUseDefaultTimeout *2203*
sslCmdFree *2205*
sslCmdGet *2205*
sslCmdInfo *2205*
sslCmdNew *2205*
sslCmdRead *2205*
sslCmdReset *2205*
sslCmdSet *2206*
sslCmdVerify *2206*
sslCmdWrite *2206*
sslCompat1RecordPerMessage *2204*
sslCompatAll *2204*
sslCompatBigRecords *2204*

sslCompatNetscapeCaDnBug *2204*
sslCompatReuseCipherBug *2205*
SslConsume() *2220*
SslContext *2192*
SslContextCreate() *2221*
SslContextDestroy() *2222*
SslContextGet_AppInt32() *2254*
SslContextGet_AppPtr() *2254*
SslContextGet_AutoFlush() *2255*
SslContextGet_BufferedReuse() *2256*
SslContextGet_CipherSuite() *2256*
SslContextGet_CipherSuiteInfo() *2257*
SslContextGet_CipherSuites() *2258*
SslContextGet_ClientCertRequest() *2258*
SslContextGet_Compat() *2259*
SslContextGet_DontSendShutdown() *2260*
SslContextGet_DontWaitForShutdown() *2260*
SslContextGet_Error() *2261*
SslContextGet_HsState() *2262*
SslContextGet_InfoCallback() *2262*
SslContextGet_InfoInterest() *2263*
SslContextGet_IoFlags() *2263*
SslContextGet_IoStruct() *2264*
SslContextGet_IoTimeout() *2265*
SslContextGet_LastAlert() *2265*
SslContextGet_LastApi() *2266*
SslContextGet_LastIo() *2266*
SslContextGet_Mode() *2267*
SslContextGet_PeerCert() *2267*
SslContextGet_PeerCommonName() *2268*
SslContextGet_ProtocolVersion() *2269*
SslContextGet_RbufSize() *2270*
SslContextGet_ReadBufPending() *2270*
SslContextGet_ReadOutstanding() *2271*
SslContextGet_ReadRecPending() *2271*
SslContextGet_ReadStreaming() *2272*
SslContextGet_SessionReused() *2272*
SslContextGet_Socket() *2273*
SslContextGet_SslSession() *2273*
SslContextGet_SslVerify() *2274*
SslContextGet_Streaming() *2275*
SslContextGet_VerifyCallback() *2275*
SslContextGet_WbufSize() *2276*
SslContextGet_WriteBufPending() *2276*
SslContextGetLong() *2223*
SslContextGetPtr() *2224*
SslContextSet_AppInt32() *2277*
SslContextSet_AppPtr() *2278*
SslContextSet_AutoFlush() *2278*
SslContextSet_BufferedReuse() *2279*
SslContextSet_CipherSuites() *2280*
SslContextSet_Compat() *2280*
SslContextSet_DontSendShutdown() *2281*
SslContextSet_DontWaitForShutdown() *2282*
SslContextSet_Error() *2282*
SslContextSet_InfoCallback() *2283*
SslContextSet_InfoInterest() *2284*
SslContextSet_IoFlags() *2285*
SslContextSet_IoStruct() *2285*
SslContextSet_IoTimeout() *2286*
SslContextSet_LastAlert() *2287*
SslContextSet_Mode() *2287*
SslContextSet_ProtocolVersion() *2288*
SslContextSet_RbufSize() *2289*
SslContextSet_ReadStreaming() *2289*
SslContextSet_Socket() *2290*
SslContextSet_SslSession() *2291*
SslContextSet_VerifyCallback() *2291*
SslContextSet_WbufSize() *2292*
SslContextSetLong() *2225*
SslContextSetPtr() *2226*
sslCs_RSA_RC4_128_MD5 *2207*
sslCs_RSA_RC4_128_SHA1 *2207*
sslCs_RSA_RC4_40_MD5 *2207*
sslCs_RSA_RC4_56_SHA1 *2207*
sslCsiAuthNULL *2206*
sslCsiAuthRsa *2206*
sslCsiCipherNull *2206*
sslCsiCipherRc4 *2206*
sslCsiDigestMd2 *2206*
sslCsiDigestMd5 *2206*
sslCsiDigestNull *2206*
sslCsiDigestSha1 *2206*
sslCsiKeyExchNull *2206*
sslCsiKeyExchRsa *2206*
sslErrBadArgument *2214*

sslErrBadDecode *2214*
sslErrBadLength *2214*
sslErrBadOption *2214*
sslErrBadPeerFinished *2214*
sslErrBadSignature *2214*
sslErrBufferTooSmall *2214*
sslErrCbAbort *2214*
sslErrCert *2215*
sslErrCertDecodeError *2215*
sslErrCsp *2215*
sslErrDivByZero *2215*
sslErrEof *2215*
sslErrExtraHandshakeData *2215*
sslErrFailed *2215*
sslErrFatalAlert *2215*
sslErrHandshakeEncoding *2215*
sslErrHandshakeProtocol *2215*
sslErrInitNotCalled *2215*
sslErrInternalError *2215*
sslErrIo *2216*
sslErrMissingProvider *2216*
sslErrNoDmem *2216*
sslErrNoMethodSet *2216*
sslErrNoModInverse *2216*
sslErrNoRandom *2216*
sslErrNotFound *2216*
sslErrNotImplemented *2216*
sslErrNullArg *2216*
sslErrOk *2216*
sslErrOutOfMemory *2216*
sslErrReadAppData *2216*
sslErrReallocStaticData *2216*
sslErrRecordError *2217*
sslErrUnexpectedRecord *2217*
sslErrUnsupportedCertType *2217*
sslErrUnsupportedSignatureType *2217*
sslErrVerifyBadSignature *2217*
sslErrVerifyCallback *2217*
sslErrVerifyConstraintViolation *2217*
sslErrVerifyNotAfter *2217*
sslErrVerifyNotBefore *2217*
sslErrVerifyNoTrustedRoot *2217*
sslErrVerifyUnknownCriticalExtension *2217*
sslErrWrongMessage *2217*
SslExtendedItem *2193*
SslExtendedItems *2194*
sslFlgInfoAlert *2209*
sslFlgInfoCert *2209*
sslFlgInfoHandshake *2209*
sslFlgInfoIo *2209*
SslFlush() *2227*
sslHsStateCert *2210*
sslHsStateCertB *2210*
sslHsStateCertReq *2210*
sslHsStateCertReqB *2210*
sslHsStateCkEx *2210*
sslHsStateCleanup *2210*
sslHsStateClientCert *2211*
sslHsStateClientHello *2211*
sslHsStateClosed *2211*
sslHsStateDone *2211*
sslHsStateFinished *2211*
sslHsStateFlush *2211*
sslHsStateGenerateKeys *2211*
sslHsStateHelloRequest *2211*
sslHsStateNone *2211*
sslHsStateReadCcs *2211*
sslHsStateReadFinished *2211*
sslHsStateReadFinishedB *2211*
sslHsStateReadFinishedC *2211*
sslHsStateServerDone *2211*
sslHsStateServerHello *2211*
sslHsStateShutdown *2211*
sslHsStateSkEx *2211*
sslHsStateSkExAnonDh *2211*
sslHsStateSkExDh *2211*
sslHsStateSkExRsa *2211*
sslHsStateStart *2211*
sslHsStateWrite *2211*
sslHsStateWriteCcs *2211*
sslHsStateWriteClose *2211*
sslHsStateWriteFlush *2211*
SslIoBuf *2195*
sslLastApiFlush *2209*
sslLastApiNone *2209*
sslLastApiOpen *2209*

sslLastApiRead *2209*
sslLastApiShutdown *2209*
sslLastApiWrite *2209*
sslLastIoNone *2210*
sslLastIoRead *2210*
sslLastIoWrite *2210*
SslLib *2196*
SslLibClose() *2228*
SslLibCreate() *2229*
SslLibDestroy() *2229*
SslLibGet_AppInt32() *2293*
SslLibGet_AppPtr() *2293*
SslLibGet_AutoFlush() *2294*
SslLibGet_BufferedReuse() *2295*
SslLibGet_CipherSuites() *2295*
SslLibGet_Compat() *2296*
SslLibGet_DontSendShutdown() *2297*
SslLibGet_DontWaitForShutdown() *2297*
SslLibGet_InfoCallback() *2298*
SslLibGet_InfoInterest() *2299*
SslLibGet_Mode() *2299*
SslLibGet_ProtocolVersion() *2300*
SslLibGet_RbufSize() *2300*
SslLibGet_ReadStreaming() *2301*
SslLibGet_VerifyCallback() *2301*
SslLibGet_WbufSize() *2302*
SslLibGetLong() *2230*
SslLibGetPtr() *2230*
SslLibName() *2231*
SslLibOpen() *2232*
SslLibSet_AppInt32() *2303*
SslLibSet_AppPtr() *2303*
SslLibSet_AutoFlush() *2304*
SslLibSet_BufferedReuse() *2305*
SslLibSet_CipherSuites() *2305*
SslLibSet_Compat() *2306*
SslLibSet_DontSendShutdown() *2307*
SslLibSet_DontWaitForShutdown() *2307*
SslLibSet_InfoCallback() *2308*
SslLibSet_InfoInterest() *2309*
SslLibSet_Mode() *2310*
SslLibSet_ProtocolVersion() *2310*
SslLibSet_RbufSize() *2311*
SslLibSet_ReadStreaming() *2312*
SslLibSet_VerifyCallback() *2312*
SslLibSet_WbufSize() *2313*
SslLibSetLong() *2232*
SslLibSetPtr() *2233*
SslLibSleep() *2234*
SslLibWake() *2234*
sslModeClear *2203*
sslModeFlush *2203*
sslModeSsl *2203*
sslModeSslClient *2203*
SslOpen() *2235*
sslOpenBufferedReuse *2202*
sslOpenDelayHandshake *2202*
sslOpenModeClear *2202*
sslOpenModeSsl *2202*
sslOpenNewConnection *2202*
sslOpenNoAutoFlush *2202*
sslOpenUseDefaultTimeout *2202*
SslPeek() *2236*
SslRead() *2237*
SslReceive() *2238*
SslSend() *2239*
SslSession *2197*
SslSocket *2198*
SslVerify *2199*
sslVerifyDone *2213*
sslVerifyExtensions *2213*
sslVerifyFindParent *2214*
sslVerifyNotAfterFindParent *2214*
sslVerifyNotBefore *2214*
sslVerifySignature *2214*
sslVersionSSLv3 *2204*
SslWrite() *2240*
standard IO functions 977
standardButtonFrame *167*
StartApplication
  and PrefGetPreferences 915
stat 753
StatBarGetIcon() *887*
StatBarSetIcon() *887*
StatGetAttribute() *888*
StatHide() *889*

StatShow() *889*
stdFont *773*, 796, 1941
StdIOPalm.h *977*
StdIOProvider.h *977*
Stereo Pan Constants *957*
StrAToI 995
StrAToI() *995*
StrCaselessCompare 995
StrCaselessCompare() *995*
StrCat 996
strcat function substitute (StrCat) *996*
StrCat() *996*
StrChr *996*
strchr function substitute (StrChr) *997*
StrChr() *996*
strcmp function substitute (StrCompare) *997*
StrCompare *997*
StrCompare() *997*
StrCompareAscii() *998*
StrCopy *999*
StrCopy() *999*
strcpy function substitute (StrCopy) *999*
StrDelocalizeNumber *999*
StrDelocalizeNumber() *999*
StrDelocalizeNumber, and launch code 16
stricmp function substitute
(StrCaselessCompare) 996
string manager 995–1014
string resource
copying 1038
string searching 256
StringMgr.h 995
StrIToA 1000
StrIToA() *1000*
StrIToH 1001
StrIToH() *1001*
StrLen 1001
strlen function substitute (StrLen) 1001
StrLen() *1001*
StrLocalizeNumber 1002
launch code 16
StrLocalizeNumber() *1002*
StrNCaselessCompare 1003
StrNCaselessCompare() *1003*
StrNCat 1004
strncat function substitute (StrNCat) 1004
StrNCat() *1003*
strncmp 1006
StrNCompare 1005
StrNCompare() *1005*
StrNCompareAscii() *1006*
StrNCopy 1006
StrNCopy() *1006*
strokes, translating 1026
StrPrintF 988, 1008
StrPrintF() *1008*
StrStr 1008
strstr function substitute (StrStr) 1008
StrStr() *1008*
StrToLower 1009
StrToLower() *1009*
Structure of a Control 167
Structure of a field 205
Structure of a Form 259
Structure of a menu 377
Structure of a Table 429
Structure of a Window 1211
structure of field object 208
StrVPrintF 990, 1010
StrVPrintF() *1010*
summary of launch codes 3, 79, 809
symbol11Font *773*
symbol7Font *773*
symbolFont *773*
sys_socket.h 1551
SysAlarmTriggeredParamType 9, 514
SysAppLaunch 1032
SysAppLaunch() *1032*
sysAppLaunchCmdAddRecord *6*
sysAppLaunchCmdAlarmTriggered *9*, 16, 511
sysAppLaunchCmdAttention *10*
sysAppLaunchCmdCardLaunch *14*
SysAppLaunchCmdCardType 15
sysAppLaunchCmdCountryChange *16*
sysAppLaunchCmdDisplayAlarm 9, *16*, 511
sysAppLaunchCmdExgAskUser *17*, 19, 1355, 1377, 1389, 1391, 2385

sysAppLaunchCmdExgGetData *19*, 19
sysAppLaunchCmdExgPreview *19*, 21, 1366, 1378, 1389, 1395
sysAppLaunchCmdExgReceiveData 21, *21*, 1366, 1372, 1376, 1378, 1395, 2385
sysAppLaunchCmdFind *22*, 1970
sysAppLaunchCmdGoTo *25*, *255*
sysAppLaunchCmdGoto 23, 53, 1356, 1376, 1388, 2386
sysAppLaunchCmdGoToURL *27*, 1402
sysAppLaunchCmdHandleSyncCallApp *28*
sysAppLaunchCmdHandleSyncCallAppType 28
sysAppLaunchCmdInitDatabase *30*
sysAppLaunchCmdLookup *31*, 2382
sysAppLaunchCmdNormalLaunch *31*
sysAppLaunchCmdNotify *32*, 82, 865
sysAppLaunchCmdOpenDB *32*
sysAppLaunchCmdPanelCalledFromApp *33*, 33, 2382
sysAppLaunchCmdReturnFromPanel 33, *33*, 2382
sysAppLaunchCmdSaveData 23, *34*
sysAppLaunchCmdSyncNotify *34*, 1397
sysAppLaunchCmdSystemLock *35*, 2382
sysAppLaunchCmdSystemReset *35*, 111
sysAppLaunchCmdTimeChange *36*
sysAppLaunchCmdURLParams *36*
SysAppLauncherDialog 427
SysAppLauncherDialog() *427*
sysAppLaunchFlagNewGlobals 26
sysAppLaunchFlagNewGlobals launch flag 37
sysAppLaunchFlagSubCal launch flag 37
sysAppLaunchFlagUIApp launch flag 37
SysBatteryInfo 2388
SysBatteryInfo() *1033*
SysBatteryInfoV20() *1034*
SysBatteryKind 901
SysBinarySearch() *1036*
SysBroadcastActionCode 1037
SysBroadcastActionCode() *1037*
SysCopyStringResource 1038
SysCopyStringResource() *1038*
SysCreateDataBaseList 1038
SysCreateDataBaseList() *1038*
SysCreatePanelList 1040
SysCreatePanelList() *1040*
SysCurAppDatabase 1041
SysCurAppDatabase() *1041*
SysDBListItemType *1031*, 1039
SysDisplayAlarmParamType 16
sysErrLibNotFound 1051, 1052
sysErrNoFreeLibSlots 1052
sysErrNoFreeRAM 1052
sysErrNoSuchFeature 713
sysErrOutOfOwnerID 1032
sysErrOutOfOwnerIDs 1038
sysErrParamErr 1032, 1038, 1053
SysErrString 1042
SysErrString() *1042*
SysEvent.h 41, 2406
SysEvtMgr.h 1015
sysExternalConnectorAttachEvent *87*, 87
sysExternalConnectorDetachEvent *87*, 87
SysFatalAlert 428
SysFatalAlert() *428*
sysFileCBtConnectPanelHelper 1600
sysFileCExpansionMgr 717
sysFileCSerialMgr 715, 716
sysFileCSmsLib 714, 2172
sysFileCSoundMgr 729
sysFileCUart328 1600
sysFileCUart328EZ 1600
sysFileCUart650 1600
sysFileCVFSMgr 717
sysFileCVirtIrComm 1600
sysFileCVirtRfComm 1600
sysFileDescStdIn 1519
SysFormPointerArrayToStrings 1042, 1383, 1384
SysFormPointerArrayToStrings() *1042*
sysFtrBtSupportsScheduledWakeup *715*
sysFtrCreator 715, 716, 718, 720, 721, 722, 727, 728, 729
sysFtrDefaultBoldFont 727
sysFtrDefaultFont 727
sysFtrIDKeyboardActive 727
sysFtrIDOEMSysHideBatteryGauge 729
sysFtrNewSerialPresent 715, 2399
sysFtrNewSerialVersion 716, 2400

sysFtrNumAccessorTrapPresent 729
sysFtrNumBacklight 718
sysFtrNumCharEncodingFlags 727, 1064
sysFtrNumCountry 727
sysFtrNumDefaultCompression 715
sysFtrNumDisplayDepth 718
sysFtrNumDmAutoBackup *718*
sysFtrNumEncoding 728, 1087, 1960, 2396
sysFtrNumEncryption 716
sysFtrNumEncryptionMaskDES 716
sysFtrNumErrorCheckLevel 726
sysFtrNumFiveWayNavVersion *720*
sysFtrNumGremlinsSupportGlobals 726
sysFtrNumHwrMiscFlags 720, 721, 722
sysFtrNumHwrMiscFlagsExt 720, 723
sysFtrNumHwrMisFlagsExt 723
sysFtrNumInputAreaFlags 718, 719
sysFtrNumIntlMgr 728, 2394
sysFtrNumLanguage 728
sysFtrNumNotifyMgrVersion 730, 860, 2408
sysFtrNumOEMCompanyID 721
sysFtrNumOEMDeviceID 721
sysFtrNumOEMHALID 721
sysFtrNumOEMROMVersion 722
sysFtrNumProcessor328 724, 725, 2391
sysFtrNumProcessorARM710A 725
sysFtrNumProcessorARM720T 725
sysFtrNumProcessorARM7TDMI 725
sysFtrNumProcessorARM920T 725
sysFtrNumProcessorARM922T 725
sysFtrNumProcessorARM925 725
sysFtrNumProcessorARM925T 725
sysFtrNumProcessorEZ 724, 725, 2391
sysFtrNumProcessorID 722, 724, 2391
sysFtrNumProcessorIs68K() *1043*
sysFtrNumProcessorIsARM() *1044*
sysFtrNumProcessorMask 724, 2391
sysFtrNumProcessorStrongARM 725
sysFtrNumProcessorSuperVZ 725
sysFtrNumProcessorVZ 725
sysFtrNumProcessorx86 725
sysFtrNumProcessorXscale 725
sysFtrNumProductID 722, 724
sysFtrNumROMVersion 722, 2382, 2385, 2390, 2438, 2440
sysFtrNumUIHardwareFlags 718, 719
sysFtrNumUIHardwareHas5Way 719
sysFtrNumUIHardwareHasJog 719
sysFtrNumUIHardwareHasJogBack 719
sysFtrNumUIHardwareHasKbd 720
sysFtrNumVendor 722
sysFtrNumWinVersion 730
sysFtrTKeyboard 727
sysFtrTOEMSys 729
SysGetOrientation 883
SysGetOrientation() *890*
SysGetOrientationTriggerState 883
SysGetOrientationTriggerState() *891*
SysGetOSVersionString 2388
SysGetOSVersionString() *1044*
SysGetRomToken 2388
SysGetROMToken() *1045*
SysGetStackInfo 2388
SysGetStackInfo() *1046*
SysGetTrapAddress() *1046*
SysGlueGetTrapAddress 1047, 1930
SysGlueTrapExists() *1955*
SysGraffitiReferenceDialog 428
SysGraffitiReferenceDialog() *428*
SysGremlins 2388
SysGremlins() *1047*
SysHandleEvent 88, 100, 107, 113, 866, 1048
SysHandleEvent() *1048*
SysInsertionSort 1048
SysInsertionSort() *1048*
SysKeyboardDialog 1050, 2384
SysKeyboardDialog() *1050*
SysKeyboardDialogV10 1050
SysKeyboardDialogV10() *1050*
SysLibFind 1051
SysLibFind() *1051*
SysLibInstall() *1051*
SysLibLoad 1052
SysLibLoad() *1052*
SysLibRemove() *1053*
sysNotifyAntennaRaisedEvent *88*

sysNotifyAppLaunchingEvent *88*
sysNotifyAppQuittingEvent *89*
SysNotifyBroadcast() *860*
SysNotifyBroadcastDeferred 380
SysNotifyBroadcastDeferred() *861*
sysNotifyBroadcasterCode 83, 859
SysNotifyBroadcastFromInterrupt() *862*
sysNotifyCardInsertedEvent *89*, 89
sysNotifyCardRemovedEvent *90*, 90
sysNotifyDBChangedEvent *92*
SysNotifyDBChangedType 93
sysNotifyDBCreatedEvent *91*
SysNotifyDBCreatedType 91
sysNotifyDBDeletedEvent *95*, 95, 625
SysNotifyDBDeletedType 95
sysNotifyDBDirtyEvent *96*
SysNotifyDBDirtyType 97
SysNotifyDBInfoType 98
sysNotifyDefaultQueueSize 859, 861
sysNotifyDeleteProtectedEvent *97*, 97
sysNotifyDeviceUnlocked *98*, 98, 107
SysNotifyDisplayChangeDetailsType 99
sysNotifyDisplayChangeEvent *99*, 1273
sysNotifyDisplayResizedEvent *100*, 885
sysNotifyEarlyWakeupEvent *100*, 100
sysNotifyErrBroadcastBusy 860
sysNotifyErrDuplicateEntry 865
sysNotifyErrEntryNotFound 867
sysNotifyErrNoStackSpace 860
sysNotifyErrQueueFull 862, 863
sysNotifyEventDequeuedEvent *101*
sysNotifyForgotPasswordEvent *101*, 101
sysNotifyGotUsersAttention *102*, 102
sysNotifyGsiDrawIndicator *102*, 102
sysNotifyHelperEvent *103*, 103, 809, 810
sysNotifyIdleTimeEvent *104*
SysNotifyInputAreaDrawingDetailsType 104
sysNotifyInputAreaDrawingEvent *104*, 104
SysNotifyInputAreaPendownDetailType 105
sysNotifyInputAreaPendownEvent *105*, 105
sysNotifyInsPtEnableEvent *106*
sysNotifyIrDASniffEvent 81
sysNotifyKeyboardDialogEvent *106*
sysNotifyLateWakeupEvent 99, *107*, 107
sysNotifyLocaleChangedEvent *108*, 108
SysNotifyLocaleChangedType 108
sysNotifyMenuCmdBarOpenEvent 66, *108*, 314, 390, 393
sysNotifyNetLibIFMediaEvent *109*, 109
SysNotifyNetLibIFMediaType 109
sysNotifyNoDatabaseID 859
sysNotifyNormalPriority 860, 864
SysNotifyParamType 32, *82*, 88, 810, 865
sysNotifyPhoneEvent 81
sysNotifyPOSEMountEvent 81
sysNotifyProcessPenStrokeEvent *110*
SysNotifyProcPtr 864, 865
SysNotifyProcPtr() *867*
SysNotifyRegister() *864*
sysNotifyResetFinishedEvent *111*, 111
sysNotifyRetryEnqueueKey *111*, 111
sysNotifySelectDay *111*, 111
SysNotifySelectDayDetailsType 112
sysNotifySleepNotifyEvent *113*, 114
sysNotifySleepRequestEvent 113, *113*
sysNotifySyncFinishEvent 35, *115*
sysNotifySyncStartEvent 35, *115*
sysNotifyTimeChangeEvent 36, *116*, 116, 1138
SysNotifyUnregister() *866*
sysNotifyVersionNum 860
sysNotifyVirtualCharHandlingEvent *116*
sysNotifyVolumeMountedEvent *117*, 117
sysNotifyVolumeUnmountedEvent *117*, 117
sysOrientationLandscape 883
sysOrientationPortrait 883
sysOrientationReverseLandscape 883
sysOrientationReversePortrait 883
sysOrientationTriggerDisabled 884
sysOrientationTriggerEnabled 884
sysOrientationUser 883
SysQSort 1053
SysQSort() *1053*
SysRandom 1055
SysRandom() *1055*
sysRandomMax 1055
SysReset 1055

SysReset() *1055*
sysResIDExtPrefs 655
sysResTExtPrefs 655
SysSetAutoOffTime 1056
SysSetAutoOffTime() *1056*
SysSetOrientation 883
SysSetOrientation() *891*
SysSetOrientationTriggerState 883
SysSetOrientationTriggerState() *892*
SysSetTrapAddress() *1056*
sysSleepAutoOff 114
sysSleepPowerButton 114
sysSleepResumed 115
sysSleepUnknown 115
SysStringByIndex 1057
SysStringByIndex() *1057*
SysTaskDelay 1058
SysTaskDelay() *1058*
System *935*
system events
    checking availability 1029
system keyboard display 1050
system() *988*
SystemMgr.h 3, 1031
SystemPreferencesChoice *899*, 899, 908
SystemPreferencesType 899
SystemResources.h 1593
SysTicksPerSecond 1058
SysTicksPerSecond() *1058*
sysTrap.... 1046, 1056
SysTraps.h 1046, 1056
SysUIAppSwitch 1058
SysUIAppSwitch() *1058*
SysUtils.h 1031

T

table functions 443–480
table objects
    fields 440
    structure 440
Table.h 429
TableAttrType *429*, 429
TableColumnAttrType *430*, 430
TableDrawItemFuncPtr 443, 460
TableDrawItemFuncType 480
TableDrawItemFuncType() *480*
TableItemPtr *432*, 432
TableItemStyleType 434
TableItemType *432*, 432
TableLoadDataFuncType 443, 460, 481
TableLoadDataFuncType() *481*
tableMaxTextItemSize 434
TablePtr *438*, 438
TableRowAttrType *438*, 438
tables
    setting load data callback 473
    setting save data callback 478
TableSaveDataFuncType 483
TableSaveDataFuncType() *483*
TableType *440*, 440
tAIB 547
taif 547
TblDrawTable() *443*
TblEditing() *444*
tblEnterEvent *73*, 73, 74, 315
TblEraseTable() *445*
tblExitEvent *73*, 74
TblFindRowData() *445*
TblFindRowID() *446*
TblGetBounds() *446*
TblGetColumnSpacing() *447*
TblGetColumnWidth() *447*
TblGetCurrentField 455
TblGetCurrentField() *448*
TblGetItemBounds() *448*
TblGetItemFont 2388
TblGetItemFont() *449*
TblGetItemInt() *449*
TblGetItemPtr() *450*
TblGetLastUsableRow() *451*
TblGetNumberOfColumns() *451*
TblGetNumberOfRows() *452*
TblGetRowData() *452*
TblGetRowHeight() *453*
TblGetRowID() *453*
TblGetSelection() *454*
TblGetTopRow() *454*

TblGlueGetColumnMasked() *1955*
TblGlueGetItemPtr() *1956*
TblGlueGetNumberOfColumns 451, 1930
TblGlueGetTopRow 454, 1930
TblGlueSetSelection 479, 1930
TblGrabFocus() *455*
TblHandleEvent 73, 74
TblHandleEvent() *456*
TblHasScrollBar() *457*
TblInsertRow() *458*
TblMarkRowInvalid() *458*
TblMarkTableInvalid() *459*
TblRedrawTable() *459*
TblReleaseFocus() *460*
TblRemoveRow() *461*
TblRowInvalid() *462*
TblRowMasked() *462*
TblRowSelectable() *463*
TblRowUsable() *463*
tblSelectEvent 73, *74*, 457
TblSelectItem() *464*
TblSetBounds() *465*
TblSetColumnEditIndicator() *465*
TblSetColumnMasked() *466*
TblSetColumnSpacing() *467*
TblSetColumnUsable() *467*
TblSetColumnWidth() *468*
TblSetCustomDrawProcedure() *468*
TblSetItemFont 2388
TblSetItemFont() *469*
TblSetItemInt() *470*
TblSetItemPtr() *471*
TblSetItemStyle() *472*
TblSetLoadDataProcedure() *473*
TblSetRowData() *474*
TblSetRowHeight() *474*
TblSetRowID() *475*
TblSetRowMasked() *475*
TblSetRowSelectable() *476*
TblSetRowStaticHeight() *477*
TblSetRowUsable() *478*
TblSetSaveDataProcedure() *478*
TblSetSelection() *479*
TblUnhighlightSelection() *480*
tblUnusableRow 451
tbmf 547
Tbmp 547
TelAppID *1661*
TelCancel() *1681*
TelCfgGetPhoneNumber() *1731*
TelCfgGetPhoneNumberType *1728*
TelCfgGetSmsCenter() *1732*
TelCfgGetSmsCenterType *1728*
TelCfgSetSmsCenter() *1734*
TelCgfGetSmsCenter 1732
TelClose() *1683*
TelClosePhoneConnection() *1683*
TelDtcCallNumber() *1764*
TelDtcCallNumberType *1760*
TelDtcCloseLine() *1765*
TelDtcReceiveData() *1767*
TelDtcReceiveDataType *1760*
TelDtcSendData() *1768*
TelDtcSendDataType *1761*
TelEmcCall() *1769*
TelEmcCloseLine() *1771*
TelEmcGetNumber() *1772*
TelEmcGetNumberCount() *1774*
TelEmcGetNumberType *1761*
TelEmcSelectNumber() *1775*
TelEmcSetNumber() *1777*
TelEmcSetNumberType *1762*
TelephonyMgr.h 2175, 2178, 2179
telErrBufferSize *1671*
telErrCodingScheme *1671*
telErrCommandFailed *1671*
telErrEntryNotFound *1671*
telErrFeatureNotSupported *1671*
telErrGenericDrvNotFound *1671*
telErrInvalidAppId *1672*
telErrInvalidDial *1672*
telErrInvalidIndex *1672*
telErrInvalidParameter *1672*
telErrInvalidString *1672*
telErrLibStillInUse *1672*
telErrLimitedCompatibility *1672*

telErrMemAllocation *1672*
telErrMsgAllocation *1672*
telErrNetworkTimeOut *1672*
telErrNoNetwork *1672*
telErrNoSIMInserted *1672*
telErrNoSpecificDrv *1672*
telErrNotInstalled *1672*
telErrorClass *1671*
telErrPassword *1673*
telErrPhoneCodeRequired *1673*
telErrPhoneComm *1673*
telErrPhoneMemAllocation *1673*
telErrPhoneMemFailure *1673*
telErrPhoneNumber *1673*
telErrPhoneReply *1673*
telErrPhoneToSIMPINRequired *1673*
telErrPIN2Required *1673*
telErrPINRequired *1673*
telErrProfileConflict *1673*
telErrPUK2Required *1673*
telErrPUKRequired *1673*
telErrResultBusyResource *1673*
telErrResultTimeOut *1674*
telErrResultUserCancel *1674*
telErrSecurity *1674*
telErrSettings *1674*
telErrSIMBusy *1674*
telErrSIMFailure *1674*
telErrSIMWrong *1674*
telErrSpcCallError *1674*
telErrSpcLineIsBusy *1674*
telErrSpcLineIsReleased *1674*
telErrSpecificDrvNotFound *1674*
telErrTooManyApps *1674*
telErrTTaskNotFound *1674*
telErrTTaskNotRunning *1674*
telErrUnavailableValue *1675*
telErrUnknown *1675*
telErrValidityPeriod *1675*
telErrValueStale *1675*
telErrVersion *1675*
TelEventType *1662*
TelGetCallState() *1684*
TelGetCallStateType *1664*
TelGetEvent() *1686*
TelGetNumberCount 1774
TelGetTelephonyEvent() *1687*
TelInfGetInformation() *1688*
TelInfGetInformationType *1665*
TelIsAvailable() *1723*
TelIsCfgServiceAvailable() *1689*
TelIsDtcServiceAvailable() *1691*
TelIsEmcServiceAvailable() *1692*
TelIsFunctionSupported() *1718*
TelIsInfServiceAvailable() *1693*
TelIsNwkServiceAvailable() *1694*
TelIsOemServiceAvailable() *1695*
TelIsPhbServiceAvailable() *1696*
TelIsPhoneConnected() *1697*
TelIsPowServiceAvailable() *1698*
TelIsServiceAvailable() *1721*
TelIsSmsServiceAvailable() *1699*
TelIsSndServiceAvailable() *1701*
TelIsSpcServiceAvailable() *1702*
TelIsStyServiceAvailable() *1703*
TelIsSupported() *1720*
TelMatchPhoneDriver() *1704*
TelNotificationType *1666*
TelNwkGetLocation() *1745*
TelNwkGetLocationType *1741*
TelNwkGetNetworkName() *1746*
TelNwkGetNetworkNameType *1742*
TelNwkGetNetworks() *1749*
TelNwkGetNetworksType *1743*
TelNwkGetNetworkType() *1750*
TelNwkGetSearchMode() *1751*
TelNwkGetSelectedNetwork 1329
TelNwkGetSelectedNetwork() *1753*
TelNwkGetSignalLevel() *1754*
TelNwkSelectNetwork() *1756*
TelNwkSetSearchMode() *1757*
TelOemCall() *1705*
TelOemCallType *1667*
TelOpen() *1706*
TelOpenPhoneConnection() *1707*
TelOpenProfile() *1708*

TelPhbAddEntry() *1863*
TelPhbDeleteEntry() *1864*
TelPhbEntryType *1858*
TelPhbGetAvailablePhonebooks() *1865*
TelPhbGetAvailablePhonebooksType *1859*
TelPhbGetEntries() *1867*
TelPhbGetEntriesType *1860*
TelPhbGetEntry() *1869*
TelPhbGetEntryCount() *1871*
TelPhbGetEntryCountType *1861*
TelPhbGetEntryMaxSizes() *1872*
TelPhbGetEntryMaxSizesType *1861*
TelPhbGetSelectedPhonebook() *1874*
TelPhbSelectPhonebook() *1875*
TelPowGetBatteryStatus() *1709*
TelPowGetPowerLevel() *1711*
TelPowSetPhonePower() *1712*
TelSendCommandString() *1713*
TelSendCommandStringType *1667*
TelServices *1676*
TelSmsDateTimeType *1799*
TelSmsDeleteMessage() *1829*
TelSmsDeleteMessageType *1800*
TelSmsDeliveryAdvancedCDMAType *1801*
TelSmsDeliveryAdvancedGSMType *1802*
TelSmsDeliveryAdvancedTDMAType *1803*
TelSmsDeliveryMessageType *1805*
TelSmsExtensionType *1808*
TelSmsGetAvailableStorage() *1830*
TelSmsGetAvailableStorageType *1809*
TelSmsGetDataMaxSize() *1832*
TelSmsGetMessageCount() *1833*
TelSmsGetMessageCountType *1810*
TelSmsGetSelectedStorage() *1835*
TelSmsGetUniquePartId() *1836*
TelSmsManualAckType *1810*
TelSmsMultiPartExtensionType *1812*
TelSmsNbsExtensionType *1813*
TelSmsReadMessage() *1838*
TelSmsReadMessages() *1840*
TelSmsReadMessagesType *1813*
TelSmsReadReport() *1842*
TelSmsReadReports() *1844*
TelSmsReadReportsType *1814*
TelSmsReadSubmittedMessage() *1846*
TelSmsReadSubmittedMessages() *1848*
TelSmsReadSubmittedMessagesType *1815*
TelSmsReportType *1815*
TelSmsSelectStorage() *1850*
TelSmsSendManualAcknowledge() *1852*
TelSmsSendMessage() *1853*
TelSmsSendMessageType *1817*
TelSmsSubmitAdvancedCDMAType *1818*
TelSmsSubmitAdvancedGSMType *1819*
TelSmsSubmitAdvancedTDMAType *1820*
TelSmsSubmitMessageType *1820*
TelSmsSubmittedMessageType *1822*
TelSmsUserExtensionType *1823*
TelSndMute() *1714*
TelSndPlayKeyTone() *1715*
TelSndPlayKeyToneType *1668*
TelSndStopKeyTone() *1717*
TelSpcAcceptCall() *1778*
TelSpcCallNumber() *1779*
TelSpcCloseLine() *1782*
TelSpcConference() *1783*
TelSpcGetCallerNumber() *1785*
TelSpcGetCallerNumberType *1763*
TelSpcHoldLine() *1787*
TelSpcPlayDTMF() *1788*
TelSpcPlayDTMFType *1763*
TelSpcRejectCall() *1789*
TelSpcRetrieveHeldLine() *1790*
TelSpcSelectLine() *1791*
TelSpcSendBurstDTMF() *1793*
TelSpcStartContinuousDTMF() *1794*
TelSpcStopContinuousDTMF() *1796*
TelStyChangeAuthenticationCode() *1735*
TelStyChangeAuthenticationType *1729*
TelStyEnterAuthenticationCode() *1737*
TelStyGetAuthenticationState() *1738*
TelUnblockNotifications() *1718*
text clipboard 215
text manager 1061–1109, 2395
text, finding with GetCharCaselessValue 601
TextMgr.h 1061

textTableItem 436, 443, 444, 460
textWithNoteTableItem 437, 443, 444, 460
tfColon *1123*
tfColon24h *1124*
tfColonAMPM *1124*
tfComma24h *1124*
tfDot *1124*
tfDot24h *1124*
tfDotAMPM *1124*
tfHours24h *1124*
tfHoursAMPM *1124*
TimAdjust 1135
TimAdjust() *1135*
TimDateTimeToSeconds 1136
TimDateTimeToSeconds() *1136*
time manager
    structures 1115
time system resource 195
time, displaying and selecting 201
TimeFormatType *1123*
TimePtr 1124
timeTableItem 437
TimeToAscii 1138
TimeToAscii() *1138*
TimeType *1124*, 1124
timeZoneStringLength 1139
TimeZoneToAscii() *1139*
TimGetSeconds 1136
TimGetSeconds() *1136*
TimGetTicks 1137
TimGetTicks() *1137*
TimSecondsToDateTime 1137
TimSecondsToDateTime() *1137*
TimSetSeconds 116, 1138
TimSetSeconds() *1138*
TimTimeZoneToUTC() *1140*
TimUTCToTimeZone() *1141*
tint 503
titles
    active area 57
    copying form title 292
transliteration 1106
translitOpCustomBase 1062
translitOpLowerCase 1062
translitOpPreprocess 1062
translitOpReserved2 1062
translitOpReserved3 1062
TranslitOpType *1062*, 1106
translitOpUpperCase 1062
TsmFepModeType *1111*
tsmFtrCreator 728
tsmFtrFlagsHasFep 728
tsmFtrNumFlags 728
TsmGetFepMode() *1112*
TsmGlueGetFepMode 1930
TsmGlueSetFepMode 1930
TsmSetFepMode() *1113*
TxtByteAttr() *1067*
TxtCaselessCompare
    and StrCaselessCompare 996, 997, 1003, 1005
TxtCaselessCompare() *1068*
TxtCharAttr() *1070*
TxtCharBounds() *1071*
TxtCharEncoding() *1072*
TxtCharIsAlNum() *1073*
TxtCharIsAlpha() *1073*
TxtCharIsCntrl() *1074*
TxtCharIsDelim() *1074*
TxtCharIsDigit() *1075*
TxtCharIsGraph() *1075*
TxtCharIsHardKey() *1076*
TxtCharIsHex() *1076*
TxtCharIsLower() *1077*
TxtCharIsPrint() *1077*
TxtCharIsPunct() *1078*
TxtCharIsRockerKey() *1078*
TxtCharIsSpace() *1079*
TxtCharIsUpper() *1079*
TxtCharIsValid() *1080*
TxtCharSize() *1080*
TxtCharWidth() *1081*
TxtCharXAttr() *1081*
TxtCompare() *1082*
TxtConvertEncoding 729
TxtConvertEncoding() *1084*
TxtConvertStateType *1062*
TxtEncodingName() *1092*

txtErrConvertOverflow 1086
txtErrTranslitOverflow 1107
txtErrTranslitOverrun 1106
txtErrTranslitUnderflow 1107
txtErrUnknownEncodingFallbackCopy 1086
txtErrUnknownTranslitOp 1106
TxtFindString 24, 1966
    and FindStrInStr 256
TxtFindString() *1093*
TxtGetChar() *1094*
TxtGetNextChar 1100
TxtGetNextChar() *1095*
TxtGetPreviousChar() *1096*
TxtGetTruncationOffset() *1097*
TxtGetWordWrapOffset() *1098*
TxtGlueByteAttr 1068, 1930
TxtGlueCaselessCompare 1069, 1931
TxtGlueCharAttr 1071, 1931
TxtGlueCharBounds 1072, 1931
TxtGlueCharEncoding 1072, 1931
TxtGlueCharIsAlNum 1073, 1931
TxtGlueCharIsAlpha 1074, 1931
TxtGlueCharIsCntrl 1074, 1931
TxtGlueCharIsDelim 1074, 1931
TxtGlueCharIsDigit 1075, 1931
TxtGlueCharIsGraph 1075, 1931
TxtGlueCharIsHex 1076, 1931
TxtGlueCharIsLower 1077, 1931
TxtGlueCharIsPrint 1077, 1931
TxtGlueCharIsPunct 1078, 1931
TxtGlueCharIsSpace 1079, 1931
TxtGlueCharIsUpper 1079, 1931
TxtGlueCharIsValid 1080, 1931
TxtGlueCharIsVirtual() *1965*
TxtGlueCharSize 1080, 1931
TxtGlueCharWidth 1931
TxtGlueCharXAttr 1082, 1931
TxtGlueCompare 1083, 1931
TxtGlueConvertEncoding() *1957*
TxtGlueEncodingName 1092, 1931
TxtGlueFindString 24, 256
TxtGlueFindString() *1966*
TxtGlueGetChar 1095, 1931
TxtGlueGetHorizEllipsisChar 2391
TxtGlueGetHorizEllipsisChar() *1967*
TxtGlueGetNextChar 1096, 1932
TxtGlueGetNumericSpaceChar 2391
TxtGlueGetNumericSpaceChar() *1967*
TxtGlueGetPreviousChar 1097, 1932
TxtGlueGetTruncationOffset 1098, 1932
TxtGlueLowerChar() *1968*
TxtGlueLowerStr() *1968*
TxtGlueMaxEncoding 1099, 1932
TxtGlueNextCharSize 1100, 1932
TxtGlueParamString 1102, 1932
TxtGluePrepFindString 1093, 1966
TxtGluePrepFindString() *1969*
TxtGluePreviousCharSize 1103, 1932
TxtGlueReplaceStr 1104, 1932
TxtGlueSetNextChar 1105, 1932
TxtGlueStrEncoding 1105, 1932
TxtGlueStripSpaces() *1970*
TxtGlueTransliterate 1107, 1932
TxtGlueTruncateString() *1971*
TxtGlueUpperChar() *1972*
TxtGlueUpperStr() *1972*
TxtGlueWordBounds 1109, 1932
TxtMaxEncoding() *1099*
TxtNameToEncoding() *1099*
TxtNextCharSize() *1100*
TxtParamString() *1101*
TxtPreviousCharSize() *1102*
TxtReplaceStr() *1103*
TxtSetNextChar() *1104*
TxtStrEncoding() *1105*
TxtTransliterate() *1106*
TxtWordBounds() *1108*

U

UDABufferSize *2356*
UDAControl() *2360*
UDAControlFunction() *2367*
UDADelete() *2361*
UDADeleteFunction() *2367*
UDAEndOfReader() *2362*
UDAExchangeReaderNew() *2366*

UDAExchangeWriterNew() *2366*
UDAFilterJoin() *2362*
UDAFilterType *2356*
UDAFlushFunction() *2367*
UDAInitiateWrite() *2363*
UDAMemoryReaderNew 2360
UDAMemoryReaderNew() *2367*
UDAMoreData() *2363*
UDAObjectType *2356*
UDARead() *2364*
UDAReaderType *2357*
UDAReadFunction() *2367*
UDAWriteFunction() *2367*
UDAWriterFlush() *2364*
UDAWriterJoin() *2365*
UDAWriterType *2358*
UIAlertFill *489*
UIAlertFrame *488*
UIBrightnessAdjust() *493*
UICaution *489*
UIColor.h 485
UIColorGetTableEntryIndex() *489*
UIColorGetTableEntryRGB() *490*
UIColorGlueGetNumTableEntries() *1973*
UIColorSetTableEntry() *491*
UIColorTableEntries *485*, 485
UICommon.h 497
UIContrastAdjust() *493*
UIControls.h 493
UIDialogFill *488*
UIDialogFrame *488*
UIFieldBackground *487*
UIFieldCaret *487*
UIFieldFepConvertedBackground *488*
UIFieldFepConvertedText *488*
UIFieldFepConvertedUnderline *489*
UIFieldFepRawBackground *488*
UIFieldFepRawText *488*
UIFieldFepUnderline *488*
UIFieldText *487*
UIFieldTextHighlightBackground *487*
UIFieldTextHighlightForeground *487*
UIFieldTextLines *487*
UIFormFill *488*
UIFormFrame *488*
UILastColorTableEntry *489*
uilibErrCurrentFocusInvalid *286*
uilibErrFormDoesNotHaveFocus *286*
uilibErrInvalidFocusObject *286*
uilibErrInvalidParam *286*
uilibErrNoNavInfoForForm *286*
uilibErrObjectFocusModeOff *286*
uilibErrObjectNotFound *286*
UIMenuFill *487*
UIMenuForeground *487*
UIMenuFrame *487*
UIMenuSelectedFill *487*
UIMenuSelectedForeground *487*
UIObjectFill *486*
UIObjectForeground *487*
UIObjectFrame *486*
UIObjectSelectedFill *487*
UIObjectSelectedForeground *487*
UIOK *489*
UIPickColor() *494*
UIPickColorStartPalette 494
UIPickColorStartRGB 494
UIPickColorStartType 494
UIResources.h 390
UIResources.r 655
UIWarning *489*
UnderlineModeType 207, *1219*, 1219
unitsEnglish *897*
unitsMetric *897*
uponPowerOff *898*
user name
    obtaining 1311

V

valid characters 1080
vchrCommand 388, 397, 398, 399, 401
vchrHardAntenna 2398
vchrMenu 314, 388, 397, 398, 399, 401, 2406
vchrRadioCoverageFail 2398
vchrRadioCoverageOK 2398
VdrvAPIType *1571*

VdrvAPIType structure 1571
VDrvClose 1582
VdrvCloseProcPtr() *1582*
VdrvConfigType *1572*, 1572
VDrvControl 1583
VdrvControlCustomProcPtr() *1585*
VdrvControlProcPtr() *1583*
VdrvCtlOpCodeEnum *1574*, 1574
VDrvCustomControl 1585
vdrvOpCodeCustom *1578*
vdrvOpCodeDisableIRDA *1577*
vdrvOpCodeDisableUART *1577*
vdrvOpCodeDisableUARTInterrupts *1578*
vdrvOpCodeEmuSetBlockingHook *1576*
vdrvOpCodeEnableIRDA *1577*
vdrvOpCodeEnableUART *1577*
vdrvOpCodeEnableUARTInterrupts *1578*
vdrvOpCodeFlushRxFIFO *1576*
vdrvOpCodeFlushTxFIFO *1576*
vdrvOpCodeGetDTRAsserted *1576*
vdrvOpCodeGetMaxRcvBlockSize *1576*
vdrvOpCodeGetOptTransmitSize *1576*
vdrvOpCodeGetUSBConfigDescriptor *1577*
vdrvOpCodeGetUSBDeviceDescriptor *1577*
vdrvOpCodeLineEnable *1577*
vdrvOpCodeNotifyBytesReadFromQ *1576*
vdrvOpCodeRcvCheckIdle *1576*
vdrvOpCodeRestoreState *1578*
vdrvOpCodeRxDisable *1577*
vdrvOpCodeRxEnable *1577*
vdrvOpCodeSaveState *1578*
vdrvOpCodeSendBufferedData *1576*
vdrvOpCodeSetCtsTimeout *1575*
vdrvOpCodeSetDTRAsserted *1576*
vdrvOpCodeSetReceiveQueue *1578*
vdrvOpCodeSetYieldPortCallback *1578*
vdrvOpCodeSetYieldPortRefCon *1578*
vdrvOpCodeSystem *1578*
vdrvOpCodeWaitForConfiguration *1576*
VDrvOpen 1586
VdrvOpenProcPtr() *1586*
VdrvOpenProcV4Ptr() *1587*
VDrvStatus 1588
VdrvStatusProcPtr() *1588*
VDrvWrite 1588
VdrvWriteProcPtr() *1588*
vdvrOpCodeClearErr *1575*
vdvrOpCodeFIFOCount *1575*
vdvrOpCodeSetBaudRate *1575*
vdvrOpCodeSetSettingsFlags *1575*
vdvrOpCodeSetSleepMode *1575*
vdvrOpCodeSetWakeupMode *1575*
vdvrOpCodeStartBreak *1575*
vdvrOpCodeStartLoopback *1576*
vdvrOpCodeStopBreak *1575*
vdvrOpCodeStopLoopback *1576*
vdvrOpCodeUserDef *1578*
vfprintf() *989*
VFS Manager 1143
VFSAnyMountParamType 117, *1144*
VFSCustomControl() *1153*
VFSDirCreate() *1154*
VFSDirEntryEnumerate() *1156*
vfsErrBadData 1151
vfsErrBadName 1151
vfsErrBufferOverflow 1151
vfsErrDirectoryNotFound 1151
vfsErrDirNotEmpty 1151
vfsErrFileAlreadyExists 1151
vfsErrFileBadRef 1152
vfsErrFileEOF 1152
vfsErrFileGeneric 1152
vfsErrFileNotFound 1152
vfsErrFilePermissionDenied 1152
vfsErrFileStillOpen 1152
vfsErrIsADirectory 1152
vfsErrNameShortened 1152
vfsErrNoFileSystem 1152
vfsErrNotADirectory 1152
vfsErrVolumeBadRef 1152
vfsErrVolumeFull 1153
vfsErrVolumeStillMounted 1153
VFSExportDatabaseToFile() *1158*
VFSExportDatabaseToFileCustom() *1159*
VFSExportProcPtr() *1207*
vfsFileAttrArchive 1150

vfsFileAttrDirectory 1150
vfsFileAttrHidden 1150
vfsFileAttrLink 1150
vfsFileAttrReadOnly 1150
vfsFileAttrSystem 1150
vfsFileAttrVolumeLabel 1150
VFSFileClose() *1161*
VFSFileCreate() *1162*
VFSFileDBGetRecord() *1163*
VFSFileDBGetResource() *1165*
VFSFileDBInfo() *1166*
VFSFileDelete() *1169*
VFSFileEOF() *1170*
VFSFileGetAttributes() *1171*
VFSFileGetDate() *1171*
VFSFileOpen() *1173*
VFSFileRead() *1174*
VFSFileReadData() *1175*
VFSFileRename() *1177*
VFSFileResize() *1178*
VFSFileSeek() *1179*
VFSFileSetAttributes() *1181*
VFSFileSetDate() *1182*
VFSFileSize() *1183*
vfsFilesystemType_AFS 1147
vfsFilesystemType_EXT2 1147
vfsFilesystemType_FAT 1148
vfsFilesystemType_FFS 1148
vfsFilesystemType_HFS 1148
vfsFilesystemType_HFSPlus 1148
vfsFilesystemType_HPFS 1148
vfsFilesystemType_MFS 1148
vfsFilesystemType_NFS 1148
vfsFilesystemType_Novell 1148
vfsFilesystemType_NTFS 1148
vfsFilesystemType_VFAT 1148
VFSFileTell() *1184*
VFSFileWrite() *1185*
vfsFtrIDDefaultFS 717
vfsFtrIDVersion 717
VFSGetDefaultDirectory() *1186*
vfsHandledStartPrc 117
vfsHandledUIAppSwitch 117
VFSImportDatabaseFromFile() *1187*
VFSImportDatabaseFromFileCustom() *1189*
VFSImportProcPtr() *1208*
VFSInstallFSLib() *1191*
vfsModeCreate 1149
vfsModeExclusive 1149
vfsModeLeaveOpen 1149
vfsModeRead 1149
vfsModeReadWrite 1149
vfsModeTruncate 1149
vfsModeVFSLayerOnly 1149
vfsModeWrite 1149
vfsMountClass_POSE 1151
vfsMountClass_Simulator 1151
vfsMountClass_SlotDriver 1151
VFSPOSEMountParamType *1145*
VFSRegisterDefaultDirectory() *1192*
VFSRemoveFSLib() *1194*
VFSSlotMountParamType *1145*
VFSUnregisterDefaultDirectory() *1194*
vfsVolumeAttrHidden 1150
vfsVolumeAttrNonRemovable *1150*
vfsVolumeAttrReadOnly 1150
vfsVolumeAttrSlotBased 1150
VFSVolumeEnumerate() *1195*
VFSVolumeFormat() *1197*
VFSVolumeGetLabel() *1199*
VFSVolumeInfo() *1200*
VFSVolumeMount() *1201*
VFSVolumeSetLabel() *1204*
VFSVolumeSize() *1205*
VFSVolumeUnmount() *1206*
Viewer application 2396
virtual character 1965
virtual driver 2399
virtual driver functions 1580
virtual driver queue functions 1589
Virtual File System Manager 1143
voltage warning threshold 1033, 1035
Volume *957*
VolumeInfoType *1146*
vsprintf (StrVPrintF) 1010
vsprintf() *990*

W

WakeupHandlerProc 1631
WakeupHandlerProcPtr 1630
WakeupHandlerProcPtr() *1631*
warningAlert *280*
Web Clipping Application Viewer 2396
whitePattern *1218*
wiCmd... constants 1927–1928
wiCmdClear *1928*
wiCmdDraw *1928*
wiCmdEnabled *1928*
WiCmdEnum 1927
wiCmdErase *1928*
wiCmdInit *1927*
wiCmdSetEnabled *1928*
wiCmdSetLocation *1928*
WinClipRectangle 1228
WinClipRectangle() *1228*
WinCopyRectangle 1229
WinCopyRectangle() *1229*
WinCreateBitmapWindow 1230
WinCreateBitmapWindow() *1230*
WinCreateOffscreenWindow 1231
WinCreateOffscreenWindow() *1231*
WinCreateWindow 1233
WinCreateWindow() *1233*
WinDeleteWindow 1234
WinDeleteWindow() *1234*
WinDirectionType 1287
winDisplayChangedEvent *75*, 885
WinDisplayToWindowPt 1235
WinDisplayToWindowPt() *1235*
window list 301
Window.h 1211
WindowFlagsType *1219*, 1219
WindowFormatType *1221*, 1231
windows 1211–1305
    active window 75
    structure 1222
WindowType *1222*
WindowType structure 1222
WinDrawBitmap 1236
WinDrawBitmap() *1236*
WinDrawChar() *1237*
WinDrawChars 1238
WinDrawChars() *1238*
WinDrawGrayLine 1239
WinDrawGrayLine() *1239*
WinDrawGrayRectangleFrame 1240
WinDrawGrayRectangleFrame() *1240*
WinDrawInvertedChars 1240
WinDrawInvertedChars() *1240*
WinDrawLine 1241
WinDrawLine() *1241*
WinDrawOperation *1224*, 1224
WinDrawPixel 1242
WinDrawPixel() *1242*
WinDrawRectangle 1242
WinDrawRectangle() *1242*
WinDrawRectangleFrame 1243
WinDrawRectangleFrame() *1243*
WinDrawTruncChars() *1244*
winEnterEvent *75*, 76, 331
winErase *1224*
WinEraseChars 1245
WinEraseChars() *1245*
WinEraseLine 1246
WinEraseLine() *1246*
WinErasePixel 1246
WinErasePixel() *1246*
WinEraseRectangle 1247
WinEraseRectangle() *1247*
WinEraseRectangleFrame 1247
WinEraseRectangleFrame() *1247*
WinEraseWindow 1248
WinEraseWindow() *1248*
winExitEvent *76*
WinFillLine 1248
WinFillLine() *1248*
WinFillRectangle 1249
WinFillRectangle() *1249*
WinGetActiveWindow 1249
WinGetActiveWindow() *1249*
WinGetBitmap() *1250*
WinGetBounds() *1250*
WinGetClip 1251

WinGetClip() *1251*
WinGetCoordinateSystem() *1251*
WinGetDisplayExtent 1252
WinGetDisplayExtent() *1252*
WinGetDisplayWindow 1252
WinGetDisplayWindow() *1252*
WinGetDrawWindow 1253
WinGetDrawWindow() *1253*
WinGetDrawWindowBounds() *1253*
WinGetFirstWindow 1253
WinGetFirstWindow() *1253*
WinGetFramesRectangle 1254
WinGetFramesRectangle() *1254*
WinGetPattern 1254
WinGetPattern() *1254*
WinGetPatternType 1255
WinGetPatternType() *1255*
WinGetPixel 1255
WinGetPixel() *1255*
WinGetPixelRGB 1256
WinGetPixelRGB() *1256*
WinGetScalingMode() *1257*
WinGetSupportedDensity() *1257*
WinGetWindowBounds 1250
WinGetWindowExtent 1258
WinGetWindowExtent() *1258*
WinGetWindowFrameRect 1259
WinGetWindowFrameRect() *1259*
WinGlueDrawChar 1237, 1932
WinGlueDrawTruncChars 1244, 1932
WinGlueGetFrameType() *1973*
WinGlueSetFrameType() *1974*
WinHandle *1226*, 1226
WinIndexToRGB 1259
WinIndexToRGB() *1259*
winInvert *1224*
WinInvertChars 1260
WinInvertChars() *1260*
WinInvertLine 1260
WinInvertLine() *1260*
WinInvertPixel 1261
WinInvertPixel() *1261*
WinInvertRectangle 1261
WinInvertRectangle() *1261*
WinInvertRectangleFrame 1262
WinInvertRectangleFrame() *1262*
WinLineType *1226*, 1226
WinLockInitType 1282
winMask *1224*
WinModal 1263
WinModal() *1263*
winOverlay *1224*
winPaint *1224*
WinPaintBitmap 1263
WinPaintBitmap() *1263*
WinPaintChar 1264
WinPaintChar() *1264*
WinPaintChars 1265
WinPaintChars() *1265*
winPaintInverse *1225*
WinPaintLine 1266
WinPaintLine() *1266*
WinPaintLines 1266
WinPaintLines() *1266*
WinPaintPixel 1267
WinPaintPixel() *1267*
WinPaintPixels 1268
WinPaintPixels() *1268*
WinPaintRectangle 1268
WinPaintRectangle() *1268*
WinPaintRectangleFrame 1269
WinPaintRectangleFrame() *1269*
WinPaintRoundedRectangleFrame() *1270*
WinPaintTiledBitmap() *1271*
WinPalette 99, 1271
WinPalette() *1271*
WinPopDrawState 1273
WinPopDrawState() *1273*
WinPtr *1227*, 1227
WinPushDrawState 1274
WinPushDrawState() *1274*
WinResetClip 1274
WinResetClip() *1274*
WinRestoreBits 1275
WinRestoreBits() *1275*
WinRGBToIndex 1275

WinRGBToIndex() *1275*
WinSaveBit() *1276*
WinSaveBits 1276
WinScaleCoord() *1277*
WinScalePoint() *1278*
WinScaleRectangle() *1279*
winScreenAllDepths *1281*
winScreenDensity *1281*
winScreenDepth *1281*
WinScreenGetAttribute() *1280*
winScreenHeight *1281*
WinScreenLock 1282
WinScreenLock() *1282*
WinScreenMode 99, 1283
WinScreenMode() *1283*
WinScreenModeOperation 1283
winScreenPixelFormat *1281*
winScreenResolutionX *1281*
winScreenResolutionY *1281*
winScreenRowBytes *1281*
WinScreenUnlock 1287
WinScreenUnlock() *1287*
winScreenWidth *1281*
WinScrollRectangle 1287
WinScrollRectangle() *1287*
WinSetActiveWindow 75, 1288
WinSetActiveWindow() *1288*
WinSetBackColor 1288
WinSetBackColor() *1288*
WinSetBackColorRGB 1289
WinSetBackColorRGB() *1289*
WinSetBounds 1290
WinSetBounds() *1290*
WinSetClip 1290
WinSetClip() *1290*
WinSetConstraintsSize() *1291*
WinSetCoordinateSystem() *1292*
WinSetDrawMode 1293
WinSetDrawMode() *1293*
WinSetDrawWindow 1293
WinSetDrawWindow() *1293*
WinSetForeColor 1294
WinSetForeColor() *1294*
WinSetForeColorRGB 1294
WinSetForeColorRGB() *1294*
WinSetPattern 1295
WinSetPattern() *1295*
WinSetPatternType 1296
WinSetPatternType() *1296*
WinSetScalingMode() *1296*
WinSetTextColor 1299
WinSetTextColor() *1299*
WinSetTextColorRGB 1300
WinSetTextColorRGB() *1300*
WinSetUnderlineMode 1301
WinSetUnderlineMode() *1301*
winSwap *1225*
WinUnscaleCoord() *1301*
WinUnscalePoint() *1302*
WinUnscaleRectangle() *1304*
WinUseTableIndexes 1272, 1273
WinValidateHandle() *1304*
WinWindowToDisplayPt 1305
WinWindowToDisplayPt() *1305*
wireless internet feature set 2396
WirelessIndicator.h 1927
word wrap 794
write callback function 1407
Write68KUnaligned32() *568*
WriteBlock 1590
WriteBlockProcPtr() *1590*
WriteByte 1591
WriteByteProcPtr() *1591*
WriteProc 1373, 1407