



C/C++ Sync Suite Reference

Palm OS® Conduit Development Kit for
Windows, Version 6.0.1

Written by Brent Gossett.

Technical assistance from Cole Goeppinger, Atul Gupta, Thomas Butler, Gerard Pallipuram, Robert Rhode, Raj Mojumder, and Jeffrey Shulman.

Copyright © 1998-2004, 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.

C/C++ Sync Suite Reference
Document Number 3012-006
May 14, 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	xxv
Related Documentation	xxvi
What this Document Contains	xxvii
Changes to This Document	xxix
Document 3012-006 for CDK 6.0.1	xxix
Document 3012-005 for CDK 6.0	xxx
Document 3012-004 for CDK 6.0	xxxi
Additional Resources	xxxiii
Conventions Used in this Document	xxxiv

Part I: Sync Manager API

1 Overview of the Sync Manager API	3
2 Schema Sync Manager API	5
Schema Sync Manager Structures and Types	6
CategoryID	8
CharEncodingType	8
ColumnID	9
DbCategoryDefn	10
DbColumnData	10
DbColumnDefn	11
DbColumnRemove	13
DbColumnName	14
DbMatchModeType	16
DbRowChangeFlags	16
DbrowData	17
DbRowResult	20
DbSchemaColumnAttrib	21
DbSchemaColumnProperty	22
DbSchemaColumnType	23
DbShareModeType	23
DbSyncMode	24

DbSyncType	25
DbTableDefn	26
RowID	27
Schema Sync Manager Constants	28
Category ID Constants	30
Category Match Modes	30
Character Encoding Types	31
Database Modification Reset Flag	33
Database Information Retrieval Option for SyncDbReadOpenDatabaseInfo()	34
Database List Search Flags	35
Database Open Modes	37
Database Share Modes	38
Maximum Name Lengths	39
Miscellaneous Schema Sync Manager Constants	39
Row Attributes	40
Row Change Flags	41
Row Data Sets	42
Structure Sizes for Schema Databases	44
Synchronization Flags	45
Synchronization Types	46
Table Column Attributes	47
Table Column Data Types	48
Schema Sync Manager Functions	50
SyncDbAddCategory	56
SyncDbAddColumns	58
SyncDbAddRowCategory	61
SyncDbAddTable	63
SyncDbCloseDatabase	65
SyncDbCreateDatabase	67
SyncDbCreateRow	70
SyncDbCreateRows	73
SyncDbDeleteAllRowsInTable	76
SyncDbDeleteDatabase	78
SyncDbDeleteRow	79

SyncDbDeleteRows.	81
SyncDbDeleteRowsInCategory.	83
SyncDbFindDatabase	85
SyncDbFindDatabaseByTypeCreator	87
SyncDbGetAllColumnValues	89
SyncDbGetCategoryCount	92
SyncDbGetCategoryDefinitionList	93
SyncDbGetChangeContext	96
SyncDbGetColumnDefinitions	99
SyncDbGetColumnIDs	101
SyncDbGetColumnPropertyValue	104
SyncDbGetColumnName	107
SyncDbGetColumnValues	110
SyncDbGetModifiedCategoryDefinitionList	113
SyncDbGetModifiedTableInfoList	116
SyncDbGetRowCategory	119
SyncDbGetRowCountInTable	122
SyncDbGetSyncMode.	124
SyncDbGetTableCount	126
SyncDbGetTableInfo	127
SyncDbGetTableInfoList	129
SyncDbGetTableSchema.	132
SyncDbIsRowInCategory	135
SyncDbModifyCategory.	138
SyncDbMoveRowsInCategory	140
SyncDbOpenDatabase	143
SyncDbPurgeAllRowsInTable	145
SyncDbReadModifiedRowInfoList	147
SyncDbReadModifiedRows	150
SyncDbReadOpenDatabaseInfo	154
SyncDbReadRow.	156
SyncDbReadRowInfo	158
SyncDbReadRowInfoList	160
SyncDbReadRows	163
SyncDbReadRowsByRowInfo	167

SyncDbReleaseStorage	170
SyncDbRemoveCategory	172
SyncDbRemoveCategoryFromAllRows	174
SyncDbRemoveColumnProperty	177
SyncDbRemoveColumns	179
SyncDbRemoveRow	181
SyncDbRemoveRowCategory	183
SyncDbRemoveRows	185
SyncDbRemoveSecretRowsInTable	187
SyncDbRemoveTable	189
SyncDbSetColumnPropertyValue.	191
SyncDbSetRowCategory	193
SyncDbWriteColumnValue	195
SyncDbWriteColumnValues	197
SyncDbWriteRow	199
SyncDbWriteRows	202
SyncReadDatabaseList	205
Schema Sync Manager Error Codes	207

3 Extended Sync Manager API	211
Extended Sync Manager Functions	212
SyncDmChangeCategory	216
SyncDmCloseDatabase	218
SyncDmCloseDatabaseEx	220
SyncDmCreateDatabase.	222
SyncDmDeleteAllResourceRecords	224
SyncDmDeleteDatabase.	226
SyncDmDeleteRecord.	228
SyncDmDeleteResourceRecord.	230
SyncDmFindDatabase	232
SyncDmFindDatabaseByTypeCreator	234
SyncDmGetDatabaseRecordCount	236
SyncDmMaxRemoteRecordSize	238
SyncDmOpenDatabase	239
SyncDmPurgeAllRecords	241

SyncDmPurgeAllRecordsInCategory	243
SyncDmPurgeDeletedRecords	245
SyncDmReadAppInfoBlock	247
SyncDmReadNextModifiedRecord	249
SyncDmReadNextModifiedRecordInCategory	253
SyncDmReadNextRecordInCategory	256
SyncDmReadOpenDatabaseInfo	259
SyncDmReadPositionXMap	261
SyncDmReadRecordByID	263
SyncDmReadRecordByIndex	267
SyncDmReadResourceRecordByIndex	270
SyncDmReadSortInfoBlock	273
SyncDmResetRecordIndex.	275
SyncDmResetSyncFlags	277
SyncDmWriteAppInfoBlock	279
SyncDmWriteRecord	281
SyncDmWriteResourceRecord	284
SyncDmWriteSortInfoBlock	287
4 Classic Sync Manager API	289
Classic Sync Manager Classes	290
CCallModuleParams	291
CDbCreateDB	293
CDbGenInfo	295
CPositionInfo	297
CRawPreferenceInfo	299
CRawRecordInfo	301
Classic Sync Manager Structures and Types	304
SyncDatabaseInfoType	305
SyncFindDbByNameParams	307
SyncFindDbByTypeCreatorParams	308
SyncReadOpenDbInfoParams	309
Classic Sync Manager Constants	310
Deprecated Constants	310
Classic Sync Manager Functions	311

SyncCallRemoteModule	315
SyncChangeCategory	319
SyncCloseDB	321
SyncCloseDBEx	322
SyncCreateDB	324
SyncDeleteAllResourceRec	326
SyncDeleteDB	327
SyncDeleteRec	328
SyncDeleteResourceRec	330
SyncFindDbByName	332
SyncFindDbByTypeCreator	334
SyncGetDBRecordCount	336
SyncMaxRemoteRecSize	337
SyncOpenDB	339
SyncPurgeAllRecs	341
SyncPurgeAllRecsInCategory	342
SyncPurgeDeletedRecs	344
SyncReadAppPreference	345
SyncReadDBAppInfoBlock	347
SyncReadDBList	349
SyncReadDBSortInfoBlock	351
SyncReadNextModifiedRec	353
SyncReadNextModifiedRecInCategory	355
SyncReadNextRecInCategory	357
SyncReadOpenDbInfo	359
SyncReadPositionXMap	361
SyncReadRecordById	364
SyncReadRecordByIndex	366
SyncReadResRecordByIndex	368
SyncResetRecordIndex	370
SyncResetSyncFlags	372
SyncWriteAppPreference	374
SyncWriteDBAppInfoBlock	376
SyncWriteDBSortInfoBlock	378
SyncWriteRec	380
SyncWriteResourceRec	382

5 Common Sync Manager API	383
Common Sync Manager Classes	384
CCallApplicationParams	385
CCardInfo	387
CDbList	389
CSyncPreference	391
CSyncProperties	393
CSystemInfo	396
CUserIDInfo	398
Common Sync Manager Structures and Types	400
CONDHANDLE	400
DBDatabaseInfo	401
Common Sync Manager Constants	405
Database Attributes	407
Database Closing Options	410
Database Information Flags	411
Database Information Retrieval Options	412
Database Search Options	413
Deprecated Constants	414
eConnType	415
eDbFlags	416
eDbOpenModes	419
eDesktopTrustStatus	421
eFirstSync	422
eMiscDbListFlags	423
eSyncPref	424
eSyncRecAttrs	425
eSyncTypes	426
Maximum Buffer Sizes	429
Miscellaneous Constants	431
Record Attributes	432
Versions of the Sync Manager API	433
Common Sync Manager Functions and Macros	434
SyncAddLogEntry	437
SyncBackupDatabase	438

SyncBackupSecurityData	442
SyncCallDeviceApplication	444
SyncGenerateBackupFileName.	448
SyncGetAPIVersion.	450
SyncGetDesktopTrustStatus	451
SyncGetHHOSVersion	452
SyncHHToHostDWord	453
SyncHHToHostWord	454
SyncHostToHHDWord	455
SyncHostToHHWord	456
SyncInstallAndBackupDatabase	457
SyncInstallDatabase	460
SyncIsDatabaseBackupNeeded.	462
SyncLoopBackTest	466
SyncReadBackupImageInfo	467
SyncReadFeature.	469
SyncReadSingleCardInfo	471
SyncReadSysDateTime	473
SyncReadSystemInfo	474
SyncReadUserID	475
SyncRebootSystem	476
SyncRegisterConduit	477
SyncRestoreSecurityData	478
SYNCROMVMAJOR	480
SYNCROMVMINOR	481
SyncUnRegisterConduit.	482
SyncWriteSysDateTime	483
SyncYieldCycles	485
Common Sync Manager Error Codes	486

Part II: Conduit APIs

6 Conduit Entry Point API	495
Conduit Entry Point API Structures	496
CfgConduitInfoType	497
ConduitRequestInfoType	499
RegistrationInfoType	500
Conduit Entry Point API Constants	502
ConduitCfgEnum	503
ConduitInfoEnum	504
Conduit Priorities	506
MFC Versions	507
Structure Sizes for Conduit Entry Points	508
Structure Versions for Conduit Entry Points	509
Conduit-Defined Entry Point API Functions	510
CfgConduit	511
ConfigureConduit	513
GetConduitInfo	515
GetConduitName	518
GetConduitVersion	520
OpenConduit	521
HotSync Manager Callback Function	523
PROGRESSFN	523
Conduit Entry Point API Error Codes	524
7 HotSync Log API	529
HotSync Log Constants	529
Activity	530
HotSync Log Functions	534
LogAddEntry	535
LogAddFormattedEntry	536
LogTestCounters	537
HotSync Log API Error Codes	538

8 Palm OS Common Language API	539
Palm OS Common Language Constants	540
Palm OS Common Language Functions	541
PalmFreeLanguage	542
PalmGetResourceVersion	543
PalmGetVersion	544
PalmLoadLanguage	545
9 Expansion Manager API	547
Expansion Manager Structures	549
ExpCardInfoType	549
Expansion Manager Constants	550
Directory Enumeration Constants	550
Hardware Capability Flags	551
Media Type Constants	552
Miscellaneous Expansion Manager Constants	553
Expansion Manager Functions	554
ExpCardInfo	555
ExpCardPresent	556
ExpSlotEnumerate	557
ExpSlotMediaType	559
Expansion Manager API Error Codes	560
10 Virtual File System Manager API	561
VFS Manager Structures and Types	562
FileInfoType	563
FileOrigin	564
FileRef	564
VFSAnyMountParamType	565
VFSSlotMountParamType	566
VolumeInfoType	567
VFS Manager Constants	569
Date Types	570
Defined File Systems	571
File and Directory Attributes	572
Invalid Reference Numbers	573

Miscellaneous Constants	574
Open Modes	575
Seek Origins	576
Versions of the VFS Manager API	577
Volume Attributes	578
Volume Mount Classes	579
Volume Format/Mount Flags	580
VFS Manager Functions	581
VFSCustomControl.	584
VFSDirCreate	586
VFSDirEntryEnumerate.	588
VFSExportDatabaseToFile	591
VFSExportDatabaseToFileEx.	593
VFSFileClose	596
VFSFileCreate	597
VFSFileDelete	599
VFSFileEOF	601
VFSFileGet	602
VFSFileGetAttributes	604
VFSFileGetDate	605
VFSFileOpen	607
VFSFilePut	609
VFSFileRead.	611
VFSFileRename	613
VFSFileResize	615
VFSFileSeek	616
VFSFileSetAttributes	618
VFSFileSetDate	619
VFSFileSize	621
VFSFileTell	622
VFSFileWrite	623
VFSGetAPIVersion	625
VFSGetDefaultDirectory	626
VFSImportDatabaseFromFile	629
VFSImportDatabaseFromFileEx	631

VFSSupport	633
VFSVolumeEnumerate	634
VFSVolumeFormat	636
VFSVolumeGetLabel	639
VFSVolumeInfo	641
VFSVolumeSetLabel	642
VFSVolumeSize	644
VFS Manager Error Codes	645

Part III: Desktop Support APIs

11 Conduit Manager API	651
Conduit Manager Structures	652
CmConduitType	653
CmConduitType2	656
CmDiscoveryInfoType	658
CM_CREATORLIST_ITEM_TYPE	659
Conduit Manager Constants	660
Conduit Information Types	661
Communications Ports	662
Conduit Manager Versions	663
Miscellaneous Constants	664
Conduit Manager Functions	665
CmConvertCreatorIDToString	676
CmConvertStringToCreatorID	677
CmGetBackupConduit	678
CmGetComPort	680
CmGetConduitByCreator	682
CmGetConduitByIndex	684
CmGetConduitCount	685
CmGetConduitCreatorID	686
CmGetCorePath	688
CmGetCreatorArgument	690
CmGetCreatorDirectory	692

CmGetCreatorFile	694
CmGetCreatorIDList	696
CmGetCreatorInfo	698
CmGetCreatorIntegrate	700
CmGetCreatorModule	701
CmGetCreatorName	703
CmGetCreatorPriority	705
CmGetCreatorRemote	707
CmGetCreatorTitle	709
CmGetCreatorType	711
CmGetCreatorUser	712
CmGetCreatorValueDword	714
CmGetCreatorValueString	716
CmGetDiscoveryInfoByIndex	718
CmGetHotSyncExecPath	719
CmGetLibVersion	721
CmGetNotifierDll	722
CmGetPCIIdentifier	724
CmGetSystemBackupConduit	725
CmGetSystemConduitByCreator	727
CmGetSystemConduitByIndex	728
CmGetSystemConduitCount	729
CmGetSystemConduitCreatorID	730
CmGetSystemCreatorDirectory	732
CmGetSystemCreatorFile	734
CmGetSystemCreatorIDList	736
CmGetSystemCreatorName	738
CmGetSystemCreatorPriority	740
CmGetSystemCreatorRemote	742
CmGetSystemCreatorTitle	744
CmGetSystemCreatorValueDword	746
CmGetSystemCreatorValueString	748
CmGetSystemDiscoveryInfoByIndex	750
CmGetSystemHotSyncExecPath	751
CmInstallConduit	754

CmInstallConduitByStruct	756
CmInstallCreator	758
CmInstallSystemConduitByStruct	760
CmInstallSystemCreator	762
CmIsCurrentUserAdmin	764
CmRemoveConduitByCreatorID	765
CmRemoveConduitByIndex	767
CmRemoveSystemConduitByCreatorID	769
CmRemoveSystemConduitByIndex	771
CmRestoreHotSyncSettings	773
CmSetBackupConduit	775
CmSetComPort	776
CmSetCorePath	777
CmSetCreatorArgument	778
CmSetCreatorDirectory	779
CmSetCreatorFile	781
CmSetCreatorInfo	783
CmSetCreatorIntegrate	784
CmSetCreatorModule	785
CmSetCreatorName	786
CmSetCreatorPriority	788
CmSetCreatorRemote	790
CmSetCreatorTitle	792
CmSetCreatorUser	794
CmSetCreatorValueDword	795
CmSetCreatorValueString	797
CmSetHotSyncExecPath	799
CmSetNotifierDll	800
CmSetPCIIdentifier	801
CmSetSystemBackupConduit	802
CmSetSystemCreatorDirectory	803
CmSetSystemCreatorFile	805
CmSetSystemCreatorName	807
CmSetSystemCreatorPriority	809
CmSetSystemCreatorRemote	811

CmSetSystemCreatorTitle	813
CmSetSystemCreatorValueDword	815
CmSetSystemCreatorValueString.	817
FmDisableCurrentUserConduitByIndex.	819
FmDisableCurrentUserConduitByPath	821
FmDisableSystemConduitByIndex	823
FmDisableSystemConduitByPath	825
FmEnableCurrentUserConduitByPath	827
FmEnableSystemConduitByPath	829
FmGetCurrentUserConduitByIndex	831
FmGetCurrentUserConduitCount	832
FmGetCurrentUserConduitFolder	833
FmGetCurrentUserDisabledConduitFolder	835
FmGetSystemConduitByIndex	837
FmGetSystemConduitCount.	838
FmGetSystemConduitFolder.	839
FmGetSystemDisabledConduitFolder.	841
Conduit Manager Error Codes	843
12 Install Conduit Manager API	847
Install Conduit Manager Structures	848
FileInstallType	848
Install Conduit Manager Functions	850
ImGetDirectory	853
ImGetDWord	855
ImGetExtension	857
ImGetMask	859
ImGetModule	860
ImGetName	862
ImGetString	864
ImGetSystemDirectory	866
ImGetSystemDWord	868
ImGetSystemExtension	870
ImGetSystemMask	872
ImGetSystemModule	873

ImGetSystemName	875
ImGetSystemString	877
ImRegister	879
ImRegisterID	880
ImRegisterSystem	881
ImRegisterSystemID	883
ImSetDirectory	884
ImSetDWord	886
ImSetExtension	888
ImSetMask	890
ImSetModule	891
ImSetName	892
ImSetString	893
ImSetSystemDirectory	895
ImSetSystemDWord	897
ImSetSystemExtension	899
ImSetSystemMask	901
ImSetSystemModule	902
ImSetSystemName	903
ImSetSystemString	904
ImUnregisterID	906
ImUnregisterSystemID	907
13 Notifier Install Manager API	909
Notifier Path and Filename	909
Notifier Install Manager Functions	910
NmFind.	911
NmFindSystem	912
NmGetByIndex	913
NmGetCount	915
NmGetSystemByIndex	916
NmGetSystemCount	918
NmRegister	919
NmRegisterSystem	920
NmRenameByIndex	921

NmRenameSystemByIndex	923
NmUnregister	925
NmUnregisterSystem	926
14 HotSync Manager API	927
HotSync Manager API Constants	928
Connection Status Flags	929
HotSync Manager API Versions	929
HSConnectionType	930
HsStatusType	931
Start Options	932
Synchronization Status Flags	934
HotSync Manager API Functions	935
HsCheckApiStatus	936
HsDisplayCustomDlg	937
HsDisplayFileLink	938
HsDisplayLog	939
HsDisplaySetupDlg	940
HsGetApiVersion	941
HsGetCommStatus	942
HsGetSyncStatus	943
HsRefreshConduitInfo	944
HsResetComm	945
HsSetAppStatus	946
HsSetCommStatus	948
HotSync Manager API Error Codes	949
15 Install Aide API	951
Install Aide API Versions	952
Install Aide Structures	953
FileInstallType	953
Install Aide Constants	955
Path Definitions	956
Structure Sizes	957
Version of the Install Aide API	957
Install Aide Functions	958

PlmGetLibVersion	961
PlmGetUserIDFromName	962
PlmGetUserNameFromID	963
PlmMoveInstallFileToHandheld	964
PlmMoveInstallFileToSlot	966
PlmSlotGetFileCount	968
PlmSlotGetFileInfo	969
PlmSlotInstallFile	971
PlmSlotMoveInstallFile	973
PlmSlotRemoveInstallFile	975
PltGetFileCount	976
PltGetFileInfo	977
PltGetFileName	979
PltGetFileTypeExtension	981
PltGetFileTypesCount.	983
PltGetInstallConduitCount	984
PltGetInstallConduitInfo	985
PltGetInstallCreatorInfo.	986
PltGetInstallFileFilter	987
PltGetInstallFileFilterForUser	988
PltGetPath	990
PltGetRegistryPath	991
Plt GetUser	992
Plt GetUserCount.	994
Plt GetUserDirectory	995
PltInstallFile	996
PltIsInstallMaskSet	997
PltIsUserProfile	998
PltRemoveInstallFile	999
PltRemoveInstallRegistry	1000
PltResetInstallMask.	1001
PltSetInstallRegistry	1002
PltSetPath	1003
Install Aide Error Codes	1004

16 User Data API	1009
User Data API Versions	1009
User Data API Constants	1011
UmUserSyncAction	1012
Palm OS Version Section and Key Names	1013
Version of the User Data API	1014
User Data API Functions	1015
UmAddUser.	1019
UmClearInstallMask	1021
UmDeleteKey	1023
UmDeleteUser	1025
UmDeleteUserPermSyncPreferences	1026
UmDeleteUserTempSyncPreferences	1028
UmGetIDFromName	1030
UmGetIDFromPath.	1032
UmGetInteger	1034
UmGetLibVersion	1036
UmGetRootDirectory	1037
UmGetString	1039
Um GetUserCount	1041
Um GetUserDirectory	1042
Um GetUserID	1044
Um GetUserName	1045
Um GetUserPassword	1047
Um GetUserPermSyncPreferences	1049
Um GetUserTempSyncPreferences	1051
Um IsInstallMaskSet	1053
Um IsUserInstalled	1055
Um IsUserNameValid	1057
Um IsUserProfile	1058
Um RemoveUserTempSyncPreferences	1059
Um SetInstallMask	1061
Um SetInteger	1063
Um SetString.	1065
Um SetUserDirectory	1067

UmSetUserInstall	1069
UmSetUserName.	1071
UmSetUserPermSyncPreferences.	1073
UmSetUserTempSyncPreferences.	1075
UmSlotGetDisplayName	1077
UmSlotGetExpMgrVersion	1079
UmSlotGetInfo.	1080
UmSlotGetInstallDirectory	1082
UmSlotGetMediaType	1084
UmSlotGetSlotCount	1086
User Data API Error Codes.	1088
17 Password Library	1093
Password Library Functions	1093
PwdVerify.	1094
18 Desktop Application Notifier API	1097
Desktop Application Notifier Constants	1097
Notification Messages	1098
Notifier-Defined API Functions.	1099
GetNotifierVersion	1100
HS_Notify.	1101
Part IV: Appendixes	
A Revision History	1105
Changes in C/C++ Sync Suite 6.0.1	1106
Sync Manager API, Version 2.5 Changes.	1106
VFS Manager API, Version 1.1 Changes	1107
File Link API Removal from HotSync Manager 6.0.1 . . .	1107
HotSync Log API Changes for HotSync Manager 6.0.1 . .	1108
User Data API, Version 4.2 Changes	1108
Palm OS Common Language API Changes in HotSync Manager 6.0.1	1108
Conduit API Changes in HotSync Manager 6.0.1	1109

HotSync Manager API Changes in HotSync	
Manager 6.0.1	1109
Changes in C/C++ Sync Suite 6.0a	1109
Changes in C/C++ Sync Suite 6.0	1110
Sync Manager API, Version 2.4 Changes.	1110
Conduit Entry Point API Changes	1111
Conduit Manager API, Version 3 Changes	1111
Install Conduit Manager API, Version 3 Changes	1114
Notifier Install Manager API, Version 3 Changes	1115
HotSync Log API Changes	1115
HotSync Manager API, Version 2 Changes.	1116
VFS Manager API, Version 1.0 Changes	1116
User Data API, Version 4.1 Changes	1116
Install Aide API, Version 4.1 Changes	1117
File Linking API Changes	1117
Palm OS Common Language API Changes	1117
Changes in C/C++ Sync Suite 4.03	1118
Sync Manager API Changes	1118
HotSync Manager API Changes	1118
User Data API, Version 4.1 Changes	1118
Changes in C/C++ Sync Suite 4.02/4.02a	1119
Expansion Manager and VFS Manager API,	
Version 1.0 Changes	1119
Sync Manager API, Version 2.3 Changes.	1119
User Data API, Version 4.0 Changes	1120
Install Aide API, Version 4.0 Changes	1121
Conduit Manager, Version 2 Changes	1122
Password Library Changes	1123
HotSync Manager API Changes	1123

B Private and Obsolete Functions	1125
Private Functions	1125
Obsolete Functions	1128
Index	1129

About This Document

The C/C++ Sync Suite is a component of the Palm OS® Conduit Development Kit (CDK) for Windows from PalmSource, Inc. It provides APIs, the C++ Generic Conduit Framework, samples, documents, and utilities to help developers create C API-based conduits that run on Windows computers. Key to the success of the Palm OS platform, conduits are software objects that exchange and synchronize data between an application running on a desktop computer and a Palm Powered™ handheld.

The C/C++ Sync Suite Reference provides you with complete reference information for all of the constants, data structures, classes, functions, and methods that you can use to develop conduits.

The sections in this introduction are:

<u>Related Documentation</u>	...	xxvi
<u>What this Document Contains</u>	...	xxvii
<u>Changes to This Document</u>	...	xxix
<u>Additional Resources</u>	...	xxxiii
<u>Conventions Used in this Document</u>	...	xxxiv

About This Document

Related Documentation

Related Documentation

The latest versions of the documents described in this section can be found at

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

The following documents are part of the CDK:

Document	Description
<u>Introduction to Conduit Development</u>	An introduction to conduits on the Windows platform. It describes how they relate to other aspects of the Palm OS platform, how they communicate with the HotSync® Manager, and how to choose an approach to conduit development. Recommended reading for developers new to conduits.
<u>C/C++ Sync Suite Companion</u>	An overview of how C API-based conduits operate and how to develop them with the C/C++ Sync Suite.
<u>C/C++ Sync Suite Reference</u>	A C API reference that contains descriptions of all conduit function calls and important data structures used to develop conduits with the C/C++ Sync Suite.
<u>COM Sync Suite Companion</u>	An overview of how COM-based conduits operate and how to develop them with the COM Sync Suite.
<u>COM Sync Suite Reference</u>	A reference for the COM Sync Suite object hierarchy, detailing each object, method, and property.
<u>Conduit Development Utilities Guide</u>	A guide to the CDK utilities that help developers create and debug conduits for Windows.

What this Document Contains

This section provides an overview of the major parts of this document and the chapters in each.

[Part I, “Sync Manager API.”](#) Describes the Sync Manager APIs that only conduits can call during a HotSync operation to synchronize one of the three different types of Palm OS databases with your desktop data source.

- [Chapter 1, “Overview of the Sync Manager API.”](#) Describes how the Sync Manager API is declared in a separate header file for each type of database it works with.
- [Chapter 2, “Schema Sync Manager API.”](#) Describes the Sync Manager functions, constants, classes, and data structures for *schema* Palm OS databases.
- [Chapter 3, “Extended Sync Manager API.”](#) Describes the Sync Manager functions, constants, classes, and data structures for *extended* Palm OS databases.
- [Chapter 4, “Classic Sync Manager API.”](#) Describes the Sync Manager functions, constants, classes, and data structures for *classic* Palm OS databases.
- [Chapter 5, “Common Sync Manager API.”](#) Describes the Sync Manager functions, constants, classes, and data structures that are either used for all types of databases or do not depend on the database type.

[Part II, “Conduit APIs.”](#) Describes the conduit entry points and the other APIs that only conduits can call during a HotSync operation.

- [Chapter 6, “Conduit Entry Point API.”](#) Describes the conduit API functions, which are functions that your conduit must implement to communicate with the HotSync Manager application.
- [Chapter 7, “HotSync Log API.”](#) Describes the HotSync log functions, which you can use to access the HotSync log.
- [Chapter 8, “Palm OS Common Language API.”](#) Describes the Palm OS Common Language API, which you can use to enable conduits to load and unload language resource DLLs.
- [Chapter 9, “Expansion Manager API.”](#) Describes the Expansion Manager functions, constants, and data

About This Document

What this Document Contains

structures, which you can use to get information about expansion cards in the handheld.

- [Chapter 10, “Virtual File System Manager API.”](#) Describes the VFS Manager functions, constants, and data structures, which you can use to access file systems on expansion cards in the handheld.

[Part III, “Desktop Support APIs.”](#) Describes the APIs that desktop applications and installers can call, usually not during a HotSync operation. However, some of these can be called by a conduit during a HotSync operation.

- [Chapter 11, “Conduit Manager API.”](#) Describes the Conduit Manager functions, constants, and data structures, which you can use to install or uninstall your conduits.
- [Chapter 12, “Install Conduit Manager API.”](#) Describes the Install Conduit Manager functions and data structures, which you can use to install or uninstall your install conduits.
- [Chapter 13, “Notifier Install Manager API.”](#) Describes the Notifier Install Manager functions, which you can use to install or uninstall your desktop notifier.
- [Chapter 14, “HotSync Manager API.”](#) Describes the HotSync Manager API functions, which you can use to control basic functions (start, stop, etc.) of HotSync Manager.
- [Chapter 15, “Install Aide API.”](#) Describes the functions that you use to install applications and databases from a desktop computer to a handheld’s main memory or any files to an expansion card.
- [Chapter 16, “User Data API.”](#) Describes the User Data functions, which you can use to access information about users who have synchronized with the desktop computer.
- [Chapter 17, “Password Library.”](#) Describes the Password Library, which you can use to verify a password.
- [Chapter 18, “Desktop Application Notifier API.”](#) Describes the API of a desktop application notifier. Your notifier implements this API so that HotSync Manager can send it messages at the beginning and end of a HotSync operation.

[Part IV, “Appendices.”](#) Provides summaries of all the C/C++ functions and error codes.

- [Appendix A, “Revision History.”](#) Details changes to APIs in the C/C++ Sync Suite in each release of the CDK for Windows.
- [Appendix B, “Private and Obsolete Functions.”](#) Summarizes the private system functions that you may see named in source code but cannot use in your conduits.

Changes to This Document

This section describes significant changes made in each version of this document. For additions and changes to C/C++ Sync Suite APIs in each version of the CDK, see [Appendix A, “Revision History,”](#) on page 1105.

- [Document 3012-006 for CDK 6.0.1](#)
- [Document 3012-005 for CDK 6.0](#)
- [Document 3012-004 for CDK 6.0](#)

Document 3012-006 for CDK 6.0.1

The significant corrections and additions in this document version are listed by chapter below:

- [Chapter 4, “Classic Sync Manager API.”](#)
 - Moved [CCallModuleParams](#) and [SyncCallRemoteModule\(\)](#) to this chapter to reflect that they work only with classic databases.
 - Added to the description of [SyncReadAppPreference\(\)](#) and [SyncWriteAppPreference\(\)](#) that these functions work with both Palm OS Cobalt and Palm OS Garnet handhelds.
- [Chapter 5, “Common Sync Manager API.”](#)
 - Noted in [CSystemInfo](#) that HotSync Manager versions 6.0 and later store the handheld’s Palm OS version number in the user data store on the desktop.
- [Chapter 7, “HotSync Log API.”](#)

About This Document

Changes to This Document

- Noted in [Activity](#) that `s1SyncFinished` and `s1SyncStarted` refer to a conduit's sync session, not the entire HotSync operation.
- [Chapter 16, “User Data API.”](#)
 - Added to [UmGetInteger\(\)](#) how to retrieve the handheld's Palm OS version number from the user data store.
- Removed the “File Linking API” chapter because this feature is no longer supported in HotSync Manager 6.0.1.
- [Appendix A, “Revision History.”](#) Added the following sections:
 - [“Changes in C/C++ Sync Suite 6.0.1”](#) on page 1106
 - [“Changes in C/C++ Sync Suite 6.0a”](#) on page 1109

Document 3012-005 for CDK 6.0

The significant changes are listed by chapter below:

- [Chapter 2, “Schema Sync Manager API.”](#)
 - Added details on the date and time data types in “[Table Column Data Types](#)” on page 48.
 - Added to [SyncDbCreateRow\(\)](#) how to create a row with no column values.
 - Added to [SyncDbCreateRows\(\)](#) that a conduit can specify row IDs only during a restore operation.
 - Added to [SyncDbRemoveTable\(\)](#) that this function cannot remove a table that contains any non-default sort indexes.
 - Added to [SyncDbWriteRows\(\)](#) how to remove a row's category membership.
- [Chapter 3, “Extended Sync Manager API.”](#)
 - Corrected list of error codes that each function can return.
 - Added to [SyncDmReadNextModifiedRecord\(\)](#) how to retrieve unread data in the cache without advancing the iteration pointer.
- [Chapter 4, “Classic Sync Manager API.”](#)

- Clarified the meaning of the [eFirstSync](#) enum values.
- Added that the result of calling [SyncCallRemoteModule\(\)](#) is indeterminate in a rare case.
- Added to [SyncReadSysDateTime\(\)](#) and [SyncWriteSysDateTime\(\)](#) the format in which the date and time is provided.
- [Chapter 10, “Virtual File System Manager API.”](#)
 - Noted that [VFSExportDatabaseToFile\(\)](#) and [VFSImportDatabaseFromFile\(\)](#) do not work with schema databases.
- [Chapter 14, “HotSync Manager API.”](#)
 - Noted that it is not necessary to call [HsRefreshConduitInfo\(\)](#) or [HsSetAppStatus\(\)](#) to make HotSync Manager versions 6.0 or later recognize a newly registered conduit.

Document 3012-004 for CDK 6.0

Most of the changes in this version document the new capabilities added in Sync Manager version 2.4 (HotSync Manager 6.0). These and other changes are listed below in chapter order:

- [Chapter 1, “Overview of the Sync Manager API.”](#) Added this chapter.
- [Chapter 2, “Schema Sync Manager API.”](#) Added this chapter.
- [Chapter 3, “Extended Sync Manager API.”](#) Added this chapter.
- [Chapter 4, “Classic Sync Manager API.”](#) Reorganized this chapter to match Sync*.h header files.
- [Chapter 5, “Common Sync Manager API.”](#) Added this chapter.
- [Chapter 7, “HotSync Log API.”](#)
 - Documented the API changes described in “[HotSync Log API Changes](#)” on page 1115.
 - Documented a [LogAddFormattedEntry\(\)](#) bug.
- Chapter 9, “File Linking API.”

About This Document

Changes to This Document

- Removed CATEGORY_REGULAR from the list of values that SubDupSubscCategory() can return. This function determines only whether the specified category name is in use as a file-link category, not whether the category exists at all.
- [Chapter 11, “Conduit Manager API.”](#)
 - Documented the API changes described in “[Conduit Manager API, Version 3 Changes](#)” on page 1111.
 - Moved the definitions of the conduit configuration entries to [Appendix A, “Configuration Entries,”](#) on page 175 in *Introduction to Conduit Development*.
- [Chapter 12, “Install Conduit Manager API.”](#)
 - Documented the API changes described in “[Install Conduit Manager API, Version 3 Changes](#)” on page 1114.
 - Changed the return value of [_ImUnregisterID\(\)](#) to 0 upon success. It erroneously stated that the value is 1.
- [Chapter 13, “Notifier Install Manager API.”](#)
 - Documented the API changes described in “[Notifier Install Manager API, Version 3 Changes](#)” on page 1115.
- [Chapter 14, “HotSync Manager API.”](#)
 - Corrected the description of [_HsResetComm\(\)](#). It stated erroneously that this function disables all transports, except local serial, when HotSync Manager is restarted.
- [Chapter 17, “Password Library.”](#)
 - Clarified the description of [_PwdVerify\(\)](#). It stated erroneously that characters in passwords on the handheld are stored as lowercase only. In fact, this is true only for Palm OS versions earlier than 4.0. In Palm OS versions 4.0 and later, passwords are case-sensitive.

Additional Resources

- Documentation

PalmSource, Inc. 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/>

- CDK Feedback

Use this email address to provide feedback on the CDK: features you would like to see, bug reports, errors in documentation, and requests for Knowledge Base articles.

cdk-feedback@palmsource.com

About This Document

Conventions Used in this Document

Conventions Used in this Document

This guide uses the following typographical conventions:

This style...	Is used for...
<code>sample</code>	Literal text such as filenames, commands, code elements such as functions, structures, and so on.
<i>sample</i>	Emphasis or to indicate a variable.
sample	Definition or first usage of a term, menu and menu item names, user-supplied text, window names in UI descriptions.
<u>sample</u>	Hypertext links.
→	Parameter is passed into a function.
←	Parameter is passed out of a function.
↔	Parameter passed in and out of a function.



Part I

Sync Manager API

The Sync Manager enables conduits to exchange data with Palm Powered™ handhelds that can communicate with a desktop computer running the HotSync® Manager application. The Sync Manager API provides functions that only conduits can call to read and write databases on the handheld during a HotSync operation.

The following chapters detail each set of Sync Manager APIs:

<u>Overview of the Sync Manager API</u>	3
<u>Schema Sync Manager API</u>	5
<u>Extended Sync Manager API</u>	211
<u>Classic Sync Manager API</u>	289
<u>Common Sync Manager API</u>	383

Overview of the Sync Manager API

The Sync Manager API is declared across four separate header files. Each header file declares the functions that access only one of the three types of databases: schema (`SyncDb.h`), extended (`SyncDm.h`), and classic (`SyncMgr.h`) databases. The fourth header file (`SyncCommon.h`) declares functions either that work with all three types of databases or that do not access databases at all; therefore it is included by each of the other three header files.

Most conduits need to access only one type of database. Therefore you should include either `SyncDb.h`, `SyncDm.h`, or `SyncMgr.h`, but not more than one of these. The distinction between extended and classic database types is much greater in the desktop Sync Manager API than it is with the handheld Data Manager API. On the handheld, the Data Manager functions work with extended databases by default, unless you include the `DmCompatibility.h` header file to work with classic databases. The Sync Manager API consists of three separate sets of functions, each for working with a different type of database.

All Sync Manager functions are available in `Sync20.dll`.

Sync Manager API Versions

The Sync Manager continues to evolve with new functions and new versions of existing functions. Each version of the Sync Manager API has a major version number and a minor version number. You can determine the version of the Sync Manager API that you are using by calling the [`SyncGetAPIVersion\(\)`](#) function.

The Sync Manager maintains backward compatibility within a major version. The Sync Manager minor version number changes when new functions are added or bugs are fixed. This document includes version information for each function.

Overview of the Sync Manager API

If your conduit depends on functions that are available only in certain versions of the Sync Manager API, you need to determine the version of the Sync Manager API with which you are dealing on a specific installation. To do so, call `SyncGetAPIVersion()`, which returns both the major version number and minor version number of the Sync Manager API on the desktop computer.

For example, if your conduit depends on a function that was added in version 2.5 of the Sync Manager API, you need to call the `SyncGetAPIVersion()` function and then verify that the major number is 2 or greater and that the minor version number is 5 or greater.

For a description of what's new in each version of the Sync Manager API, see "[What's New in the Palm OS CDK](#)" on page 1 in the *Introduction to Conduit Development*.

Schema Sync Manager API

The schema Sync Manager API consists of only those functions that work with [schema databases](#). Other Sync Manager functions that do not depend on the database type are described in [Chapter 5, “Common Sync Manager API,”](#) on page 383.

This chapter has the following sections:

Schema Sync Manager Structures and Types	6
Schema Sync Manager Constants	28
Schema Sync Manager Functions	50
Schema Sync Manager Error Codes	207

All Sync Manager functions are available in Sync20.dll. The API described in this chapter is declared in SyncDb.h.

Schema Sync Manager API

Schema Sync Manager Structures and Types

Schema Sync Manager Structures and Types

This section describes the following data structures and data types that you use only with the functions described in “[Schema Sync Manager Functions](#)” on page 50. Additional structures and types used by Sync Manager functions that work on schema as well as extended and classic databases are described in “[Common Sync Manager Structures and Types](#)” on page 400.

CategoryID	Defines a type for a unique category ID in a schema database.
CharEncodingType	Defines the type of character encoding (ASCII, Palm Latin, Shift JIS, and so on) used in DBDatabaseInfo.encoding.
ColumnID	Defines a column ID in a schema database.
DbCategoryDefn	Defines information about a category in a schema database.
DbColumnData	Defines a generic type pointer used for column data in a DbColumnValue structure.
DbColumnDefn	Defines the properties of a single column in a table.
DbColumnRemove	Specifies a column to remove from a table and receives an error code upon return.
DbColumnValue	Defines a table column value. You use this structure when reading or writing one or more column values in a database row.
DbMatchModeType	Defines how a row's category membership should match a supplied set of categories.
DbRowChangeFlags	Define values for how or whether a row has been modified since the last HotSync operation.
DbrowData	Defines all of the data in a table row, including column values and metadata about a row.
DbRowResult	Specifies a row to be deleted or removed and receives an error code upon return.

DbSchemaColumnAttrib	Defines the type for a combination of table column attributes.
DbSchemaColumnProperty	Defines a column property ID.
DbSchemaColumnType	Defines the data type of a table column.
DbShareModeType	Defines a type for schema database share modes.
DbSyncMode	Receives the type of synchronization that the Sync Manager determines should occur with the current desktop for each type of sync atom in a schema database.
DbSyncType	Receives the type of synchronization that the Sync Manager determines should occur for a sync atom in a schema database.
DbTableDefn	Defines a table in a schema database.
RowID	Defines a type for a unique row ID in a schema database.

Schema Sync Manager API

CategoryID

CategoryID Typedef

Purpose	Defines a type for a unique category ID in a schema database.
Declared In	SyncDb.h
Prototype	typedef SInt32 CategoryID
Comments	<p>Each schema database can have up to 255 categories and each row can belong to multiple categories, up to the same maximum of 255.</p> <p>Category IDs are assigned by the Category Manager on the handheld and are unique within a database. The Category Manager always assigns positive values. Therefore, because CategoryID on the desktop is a signed integer, a desktop application can assign a negative value to a category ID and be assured that it cannot conflict with one created on the handheld since the last HotSync operation. Then a conduit can replace it with the positive category ID assigned by the handheld during the next HotSync operation.</p>
Compatibility	Sync Manager version: 2.4 or later. Palm OS® version: Palm OS Cobalt, version 6.0 or later.
See Also	“Category ID Constants” on page 30

CharEncodingType Typedef

Purpose	Defines the type of character encoding (ASCII, Palm Latin, Shift JIS, and so on) used in DBDatabaseInfo .encoding.
Declared In	SyncDb.h
Prototype	typedef UInt16 CharEncodingType
Comments	A variable of this type can have any of the values defined in TextMgr.h provided in the Software Development Kit for Palm OS Cobalt. The SyncDb.h file provided in the CDK declares many of these same values, as described in “ Character Encoding Types ” on page 31.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	DBDatabaseInfo , “ Character Encoding Types ” on page 31

ColumnID Typedef

Purpose	Defines a column ID in a schema database.
Declared In	SyncDb.h
Prototype	<code>typedef UInt32 ColumnID</code>
Comments	A column ID uniquely identifies a column in a table in a schema database. A column defines a common property set for all column values across multiple rows. A collection of all column property sets defines a schema for all the rows in a table. Handheld applications or conduits are responsible for assigning column IDs, which must be unique within a table.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.

Schema Sync Manager API

DbCategoryDefn

DbCategoryDefn Struct

Purpose	Defines information about a category in a schema database.
Declared In	SyncDb.h
Prototype	<pre>typedef struct DbCategoryDefn { CategoryID categoryID; char name[catCategoryNameLength]; } DbCategoryDefn;</pre>
Fields	categoryID Defines the unique identifier of a category as a CategoryID . name Defines the null-terminated category name as a character array. This name is unique among all categories in a database. The maximum length of a category name, including the NULL terminator, is defined by the catCategoryNameLength constant in “ Maximum Name Lengths ” on page 39.
Comments	The SIZEOF_DB_CATEGORY_DEFN constant defines the size of a DbCategoryDefn structure. “ Structure Sizes for Schema Databases ” on page 44 defines this and other size constants provided for your convenience.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	CategoryID

DbColumnData Typedef

Purpose	Defines a generic type pointer used for column data in a DbColumnName structure.
Declared In	SyncDb.h
Prototype	<pre>typedef void DbColumnData</pre>
Comments	When receiving a column value, this refers to a memory location large enough to receive a value for that data type.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.

DbColumnDefn Struct

Purpose	Defines the properties of a single column in a table.
Declared In	<code>SyncDb.h</code>
Prototype	<pre>typedef struct DbColumnDefn { ColumnID columnID; UInt32 columnMaxSize; DbSchemaColumnType columnType; DbSchemaColumnAttrib columnAttributes; char columnName [dmColumnNameLength]; UInt32 errCode; } DbColumnDefn</pre>
Fields	<p>columnID Defines a unique column identifier of type ColumnID.</p> <p>columnMaxSize Defines a size specification for the column data, in bytes. For variable-length string vectors, it specifies the size upper-bound and for fixed-length strings, the actual size. For vectors, it specifies the upper-bound in terms of byte count.</p> <p>columnType Defines the data type of the column as a DbSchemaColumnType value. This field can have one of the values defined in “Table Column Data Types” on page 48.</p> <p>columnAttributes Defines the attributes of the column as a DbSchemaColumnAttrib value. This field can have one or more of the values defined in “Table Column Attributes” on page 47.</p> <p>columnName Defines the null-terminated name of the column as a character array. This name is unique among all columns in a table. The maximum length of a column name, including the NULL terminator, is defined by the <code>dmColumnNameLength</code> constant in “Maximum Name Lengths” on page 39.</p> <p>errCode If successful, receives <code>SYNCERR_NONE</code>. If unsuccessful when adding a column or table definition, receives one of the following error code values:</p>

Schema Sync Manager API

DbColumnDefn

SYNCERR_COLUMNID_ALREADY_EXISTS
The column ID specified in `columnID` already exists.

SYNCERR_INVALID_COLUMNNAME
The column name specified in `columnName` is invalid.

SYNCERR_INVALID_COLUMNSIZE
The column data size specified in `columnMaxSize` for a fixed-sized column is invalid.

SYNCERR_INVALID_COLUMNSPEC
One or more of the values specified in `columnAttributes` are invalid.

SYNCERR_INVALID_COLUMNTYPE
The column data type specified in `columnType` is not a defined type.

SYNCERR_NAME_ALREADY_EXISTS
The column name specified in `columnName` already exists.

If unsuccessful when reading a column definition, receives one of the following error code values:

SYNCERR_INVALID_COLUMNID
The column ID specified in `columnID` is invalid.

Comments You work with these structures in arrays when adding columns and getting column definitions with [SyncDbAddColumns\(\)](#) and [SyncDbGetColumnDefinitions\(\)](#). A table definition contains an array of these structures; see “[DbTableDefn](#)” on page 26.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

DbColumnRemove Struct

Purpose Specifies a column to remove from a table and receives an error code upon return.

Declared In SyncDb.h

Prototype

```
typedef struct DbColumnRemove {  
    ColumnID columnID;  
    UInt32 errCode;  
} DbColumnRemove
```

Fields columnID
Specifies a unique column identifier of type [ColumnID](#).

errCode

If successful, receives SYNCERR_NONE.

If unsuccessful, receives the following error code value:

SYNCERR_INVALID_COLUMNID

The specified column ID is invalid.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncDbRemoveColumns \(\)](#)

Schema Sync Manager API

DbColumnValue

DbColumnValue Struct

Purpose	Defines a table column value. You use this structure when reading or writing one or more column values in a database row.
Declared In	<code>SyncDb.h</code>
Prototype	<pre>typedef struct DbColumnValue { ColumnID columnID; UInt32 dataSize; DbColumnData *columnData; UInt32 errCode; } DbColumnValue</pre>
Fields	<p>columnID Defines a unique column identifier of type ColumnID.</p> <p>dataSize Defines the size, in bytes, of the column data being read or written. For variable-length string types, it specifies the actual size to be read or written. For vectors, it specifies the actual byte count to be read or written.</p> <p>When reading a column value, this field specifies the number of bytes in the array that <code>*columnData</code> points to; upon return it receives either the actual size of the column value on the handheld or the number of bytes received in the <code>*columnData</code> array. The value received by this field is valid only if the <code>errCode</code> field receives <code>SYNCERR_NONE</code> or <code>SYNCERR_MORE</code>. When writing a column value, the caller specifies the number of bytes in the <code>*columnData</code> array.</p> <p>columnData Defines a pointer to the raw data of the column value.</p> <p>errCode If successful, receives one of the following values:</p> <ul style="list-style-type: none">SYNCERR_NONE The read or write operation is complete.SYNCERR_MORE When reading, the <code>*columnData</code> buffer is too small to hold all the column value. Allocate at least as much

memory as indicated by the value that the `dataSize` field receives and call the same function again.

If unsuccessful, receives one of the following error code values:

`SYNCERR_INVALID_COLUMNSIZE`

When writing, the column data size specified in `dataSize` for a fixed-sized column is invalid.

`SYNCERR_OPERATION_ABORTED`

When writing, the operation aborted and did not write the remaining column values.

`SYNCERR_INVALID_COLUMNID`

When reading or writing, the column ID specified in the `columnID` field is invalid.

`SYNCERR_NO_DATA`

When reading, no column value exists.

Comments If the `errCode` field receives `SYNCERR_MORE` when reading, the `dataSize` field receives the actual size of the column value. Reallocate the `*columnData` buffer to this size and call the function again to retrieve the entire column value. The schema Sync Manager copies the data only when the buffer is large enough to hold all of the column value.

However, functions that retrieve data in a `DbRowData` structure, which includes an array of `DbColumnValue` structures, allocate memory themselves. Therefore calls to these functions never pass back `SYNCERR_MORE` in the `errCode` field, and thus do not require you call them more than once to retrieve column values.

The `SIZEOF_DB_COLUMN_VALUE` constant defines the size of a `DbColumnValue` structure. “[Structure Sizes for Schema Databases](#)” on page 44 defines this and other size constants provided for your convenience.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [`DbRowData`](#), [`SyncDbCreateRow\(\)`](#),
[`SyncDbGetColumnValues\(\)`](#),
[`SyncDbGetAllColumnValues\(\)`](#), [`SyncDbWriteRow\(\)`](#),
[`SyncDbWriteColumnValues\(\)`](#)

Schema Sync Manager API

DbMatchModeType

DbMatchModeType Typedef

Purpose	Defines how a row's category membership should match a supplied set of categories.
Declared In	<code>SyncDb.h</code>
Prototype	<code>typedef UInt32 DbMatchModeType</code>
Comments	A variable of this type can have one of the values defined in “ Category Match Modes ” on page 30.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	“ Category Match Modes ” on page 30

DbRowChangeFlags Typedef

Purpose	Define values for how or whether a row has been modified since the last HotSync operation.
Declared In	<code>SyncDb.h</code>
Prototype	<code>typedef UInt16 DbRowChangeFlags</code>
Comments	A variable of this type can have a valid combination of one or more of the values defined in “ Row Change Flags ” on page 41. This type is used by the <code>changeFlags</code> field of the DbRowData structure.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	“ Row Change Flags ” on page 41

DbRowData Struct

Purpose Defines all of the data in a table row, including column values and metadata about a row.

Declared In SyncDb.h

Prototype

```
typedef struct DbRowData {
    UInt32 dataSetRetrieved;
    UInt32 errCode;
    RowID rowID;
    UInt16 attributes;
    const char *pTableName;
    DbRowChangeFlags changeFlags;
    UInt32 numCategories;
    CategoryID *pCategoryIDList;
    UInt32 numColumns;
    DbColumnValue *pColumnValues;
    void *pUpdate;
} DbRowData
```

Fields dataSetRetrieved

Defines the type of data or metadata that has been retrieved (or is to be retrieved) for this table row. Specify one of the values defined in “[Row Data Sets](#)” on page 42 when calling a read function. Upon return, this field receives a value indicating what was successfully retrieved.

Present in all data sets defined by “[Row Data Sets](#)” on page 42, except for dbRowEmptyDataSet.

errCode

Receives an error code indicating whether the operation was successful. If an operation fails, then this field indicates an error for the data set after the last one that succeeded, if any. For a list of possible error codes, see the description of each function that uses the DbRowData structure.

Present in all data sets defined by “[Row Data Sets](#)” on page 42, except for dbRowEmptyDataSet.

rowID

Defines a unique row identifier of type [RowID](#).

Present in all data sets defined by “[Row Data Sets](#)” on page 42, except for dbRowEmptyDataSet.

Schema Sync Manager API

DbRowData

attributes

Defines the attributes of the row. This field can have a valid combination of one or more of the values defined in “[Row Attributes](#)” on page 40. The only valid attributes are dbRecAttrSecret and dbRecAttrReadOnly.

Present in the dbRowInfoDataSet data set.

pTableName

Receives a pointer to a read-only char buffer containing the table name that the row belongs to.

Present in the dbRowInfoDataSet data set.

changeFlags

Receives a valid combination of the values defined in “[Row Change Flags](#)” on page 41. These flags indicate how or whether a row has been modified since the last HotSync operation.

Present in the dbRowInfoDataSet data set.

numCategories

Defines the number of categories that this row belongs to. When writing, specify a value less than or equal to the number of IDs in the *pCategoryIDList array. Only this number of category IDs are written; any additional ones in the array are ignored.

Present in the dbRowCategoriesDataSet data set.

pCategoryIDList

Defines an array of [CategoryID](#) values that indicates which categories this row belongs to.

Present in the dbRowCategoriesDataSet data set.

numColumns

Defines the number of column values in this row. When writing, specify a value less than or equal to the number of column values in the *pColumnValues array. Only this number of column values are written; any additional ones in the array are ignored.

Present in the dbRowColumnValuesDataSet data set.

pColumnValues

Defines an array of [DbColumnValue](#) structures contain the column values for this row.

Present in the dbRowColumnValuesDataSet data set.

pUpdate

Reserved for future use.

Comments

This structure represents all of the data associated with a table row. The read functions that use this structure allocate memory for you. To further aid you, the Sync Manager defines several row data sets described in “[Row Data Sets](#)” on page 42. These sets allow you to specify the sets of fields in this structure that you want to read. The Sync Manager retrieves these data sets in the following order:

1. dbRowEmptyDataSet
2. dbRowInfoDataSet
3. dbRowCategoriesDataSet
4. dbRowColumnInfoDataSet
5. dbRowColumnValuesDataSet
6. dbRowAllDataSet

Each time it *successfully* retrieves a data set, the Sync Manager sets the dataSetRetrieved field to the value of that data set and the errCode value to SYNCERR_NONE. If it fails, the dataSetRetrieved field indicates the last successfully retrieved data set and errCode indicates the error encountered while retrieving the next data set. When the function returns, you can use these fields and the order in which data sets are retrieved to determine which data sets contain valid data and on which one the function failed.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

“[Row Data Sets](#)” on page 42, [SyncDbReadRowInfo\(\)](#),
[SyncDbReadRowInfoList\(\)](#),
[SyncDbReadModifiedRowInfoList\(\)](#), [SyncDbReadRow\(\)](#),
[SyncDbReadRows\(\)](#), [SyncDbReadModifiedRows\(\)](#),
[SyncDbReadRowsByRowInfo\(\)](#), [SyncDbCreateRows\(\)](#),
[SyncDbWriteRows\(\)](#)

Schema Sync Manager API

DbRowResult

DbRowResult Struct

Purpose	Specifies a row to be deleted or removed and receives an error code upon return.
Declared In	<code>SyncDb.h</code>
Prototype	<pre>typedef struct DbRowResult { RowID rowID; UInt32 errCode; } DbRowResult</pre>
Fields	rowID Specifies the unique row identifier of type RowID for a row to mark as deleted or to remove. errCode Receives an error code upon return to indicate whether the operation succeeded. If successful, receives SYNCERR_NONE. If unsuccessful, receives the following error code value: SYNCERR_NOT_FOUND The specified row is not present in the database.
Comments	You work with these structures in arrays when marking rows as deleted with SyncDbDeleteRows () and removing them with SyncDbRemoveRows () .
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncDbDeleteRows () , SyncDbRemoveRows ()

DbSchemaColumnAttrib Typedef

Purpose	Defines the type for a combination of table column attributes.
Declared In	SyncDb.h
Prototype	<code>typedef UInt8 DbSchemaColumnAttrib</code>
Comments	A variable of this type can have one or more of the values defined in “ Table Column Attributes ” on page 47.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	“ Table Column Attributes ” on page 47

Schema Sync Manager API

DbSchemaColumnProperty

DbSchemaColumnProperty Typedef

Purpose	Defines a column property ID.
Declared In	<code>SyncDb.h</code>
Prototype	<code>typedef UInt8 DbSchemaColumnProperty</code>
Comments	A property ID uniquely identifies each column property in a table. The range of property IDs from 0x00 to <code>dbColumnPropertyUpperBound</code> are reserved for built-in properties, which a conduit cannot remove or modify, like property ID, name, data type, size, and attributes. A conduit can assign property IDs to custom properties as long as the ID is greater than <code>dbColumnPropertyUpperBound</code> . The Sync Manager retrieves values of built-in properties as part of column definitions. Therefore to read the values of built-in properties, call <code>SyncDbGetColumnDefinitions()</code> or <code>SyncDbGetTableSchema()</code> which pass back <code>DbColumnDefn</code> or <code>DbTableDefn</code> structures, respectively, that receive the values of built-in properties. The schema Sync Manager API also includes functions to retrieve, set, or remove custom properties.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	<code>SyncDbGetColumnDefinitions()</code> , <code>SyncDbGetTableSchema()</code> , <code>SyncDbGetColumnPropertyValue()</code> , <code>SyncDbSetColumnPropertyValue()</code> , <code>SyncDbRemoveColumnProperty()</code>

DbSchemaColumnType Typedef

Purpose	Defines the data type of a table column.
Declared In	<code>SyncDb.h</code>
Prototype	<code>typedef UInt8 DbSchemaColumnType</code>
Comments	A variable of this type can have one of the values defined in “ Table Column Data Types ” on page 48.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	“Table Column Data Types” on page 48

DbShareModeType Typedef

Purpose	Defines a type for schema database share modes.
Declared In	<code>SyncDb.h</code>
Prototype	<code>typedef UInt16 DbShareModeType</code>
Comments	A variable of this type can have one of the values defined in “ Database Share Modes ” on page 38.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	“Database Share Modes” on page 38

Schema Sync Manager API

DbSyncMode

DbSyncMode Struct

Purpose	Receives the type of synchronization that the Sync Manager determines should occur with the current desktop for each type of sync atoms in a schema database.
Declared In	<code>SyncDb.h</code>
Prototype	<pre>typedef struct DbSyncMode { DbSyncType schema; DbSyncType category; DbSyncType row; } DbSyncMode</pre>
Fields	schema Receives the synchronization type for schema definitions (DbTableDefn). category Receives the synchronization type for category definitions (DbCategoryDefn). row Receives the synchronization type for rows (DbRowData).
Comments	SyncDbGetSyncMode() initializes this structure and passes it back via the <i>pSyncMode</i> parameter. Each field receives a DbSyncType structure that specifies whether the synchronization type is fast or slow for that sync atom. The DbSyncType structure further indicates whether any sync atoms of each type have been purged since the last HotSync operation with this desktop.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncDbGetSyncMode() , DbSyncType

DbSyncType Struct

Purpose Receives the type of synchronization that the Sync Manager determines should occur for a [sync atoms](#) in a schema database.

Declared In SyncDb.h

Prototype

```
typedef struct DbSyncType {  
    UInt8 type;  
    UInt8 flags;  
} DbSyncType
```

Fields

type Receives a constant that indicates whether the conduit should handle the current HotSync operation as a slow sync or a fast sync for all instances of this sync atom. This field has one of the values defined in “[Synchronization Types](#)” on page 46.

flags

Receives a constant that indicates whether deleted instances of this sync atom have been purged from the handheld since the last HotSync operation with this desktop. This field takes one of the values defined in “[Synchronization Flags](#)” on page 45.

Comments

When you call [SyncDbGetSyncMode \(\)](#), each field of the [DbSyncMode](#) structure receives a DbSyncType structure that receives a sync atom’s synchronization type. The flags field further qualifies the synchronization type in the type field. The flags field qualifies the synchronization type and has the following values depending on the value of the type field:

- flags can be dbDeletesPurged only when type is dbFastSync.
- flags is always set to dbNoFlag if type is set to dbSlowSync.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[SyncDbGetSyncMode \(\)](#), [DbSyncMode](#)

Schema Sync Manager API

DbTableDefn

DbTableDefn Struct

Purpose Defines a table in a schema database.

Declared In SyncDb.h

Prototype

```
typedef struct DbTableDefn {  
    UInt32 numColumns;  
    char name[dmTableNameLength];  
    DbColumnDefn *columnList;  
} DbTableDefn
```

Fields numColumns
Defines the number of column definitions in a table, which is also the number of elements in the columnList array.

name
Defines the null-terminated name of a table as a character array. The maximum length of a table name, including the NULL terminator, is defined by the dmTableNameLength constant in “[Maximum Name Lengths](#)” on page 39.

columnList
Defines an array of [DbColumnDefn](#) structures that define the column ID, maximum size, data type, attributes, and name of each column in a table. The number of elements in this array is defined by the numColumns field.

Comments Table definitions enable a schema database to describe itself to data consumers and suppliers. You use this table definition structure when creating a database with either [SyncDbCreateDatabase\(\)](#), when adding a new table to a database with [SyncDbAddTable\(\)](#), and when querying a database table for schema information with [SyncDbGetTableSchema\(\)](#), [SyncDbGetTableInfo\(\)](#), [SyncDbGetTableInfoList\(\)](#), and [SyncDbGetModifiedTableInfoList\(\)](#).

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [DbColumnDefn](#), [SyncDbCreateDatabase\(\)](#),
[SyncDbAddTable\(\)](#), [SyncDbGetTableSchema\(\)](#),
[SyncDbGetTableInfo\(\)](#), [SyncDbGetTableInfoList\(\)](#),
[SyncDbGetModifiedTableInfoList\(\)](#)

RowID Typedef

Purpose	Defines a type for a unique row ID in a schema database.
Declared In	SyncDb.h
Prototype	<code>typedef UInt32 RowID</code>
Comments	Row IDs are unique within a schema database. Normally, the Data Manager on the handheld assigns row IDs—for example, when a conduit calls SyncDbCreateRow() , this function passes back the ID of the newly created row. An exception is a restore operation (desktop overwrites handheld): a conduit first either deletes the database and recreates it, or purges all deleted rows and creates new rows. To create new rows in this case, a conduit can call SyncDbCreateRow() or SyncDbCreateRows() and pass in row ID values.
See Also	SyncDbCreateRow() , SyncDbCreateRows()
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.

Schema Sync Manager API

Schema Sync Manager Constants

Schema Sync Manager Constants

This section describes the following groups of preprocessor constants that you use only with the functions described in “[Schema Sync Manager Functions](#)” on page 50. Additional constants used by schema as well as extended and classic Sync Manager functions are described in “[Common Sync Manager Constants](#)” on page 405.

Category ID Constants	Define special category ID values for “Unfiled” table rows for use with specific functions.
Category Match Modes	Define how a row’s category membership should match a supplied set of categories.
Character Encoding Types	Define the character encoding types for the encoding field of the DBDatabaseInfo structure.
Database Modification Reset Flag	Indicates that the Sync Manager is to reset the modification state of a schema database when it is closed.
Database Information Retrieval Option for SyncDbReadOpenDatabaseInfo()	Indicates that SyncDbReadOpenDatabaseInfo() retrieves extra information about a schema database.
Database List Search Flags	Indicate what types of databases to search for when reading a list of databases with SyncReadDatabaseList().
Database Open Modes	Defines the access modes in which a schema database can be opened.
Database Share Modes	Define the shared access modes, which control how others can access a database that your application has opened.
Maximum Name Lengths	Define the maximum length of category, column, and table names for schema databases.
Miscellaneous Schema Sync Manager Constants	Define constants used with various schema Sync Manager functions.

<u>Row Attributes</u>	Define attributes of rows in a schema database when passed back in the attributes field in a DbRowData structure.
<u>Row Change Flags</u>	Define values for the changeFlags field of the DbRowData structure that indicate how or whether a row has been modified since the last HotSync operation.
<u>Row Data Sets</u>	Indicate the defined sets of data or metadata that have been retrieved (or are to be retrieved) for a table row. These are the values of the DbRowData.dataSetRetrieved field.
<u>Structure Sizes for Schema Databases</u>	Define the sizes of various structures used by schema Sync Manager functions.
<u>Synchronization Flags</u>	Indicate whether a deleted sync atom in a schema database has already been purged from the handheld.
<u>Synchronization Types</u>	Indicate whether a conduit needs to perform a fast, slow, or no synchronization on a given sync atom in a schema database.
<u>Table Column Attributes</u>	Define the attributes of a table column.
<u>Table Column Data Types</u>	Define the permissible data types that columns can be defined as in a table.

Schema Sync Manager API

Category ID Constants

Category ID Constants

Purpose	Define special category ID values for “Unfiled” table rows for use with specific functions.
Declared In	SyncDb.h
Constants	<pre>#define catIDAll ((CategoryID) 0x01) When specified as the only category ID value in SyncDbRemoveRowCategory(), this value sets the category membership of the specified row to “Unfiled”. Do not use this value with any other function. #define catIDUnfiled ((CategoryID) 0x00) When specified in SyncDbWriteRow() or SyncDbWriteRows(), this value sets the category membership of the specified rows to “Unfiled”. Do not use this value with any other function.</pre>
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	CategoryID , SyncDbRemoveRowCategory() , SyncDbWriteRow() , SyncDbWriteRows()

Category Match Modes

Purpose	Define how a row’s category membership should match a supplied set of categories.
Declared In	SyncDb.h
Constants	<pre>#define DbMatchAll ((DbMatchModeType) 0x02) (AND) Match rows that are in <i>all</i> of the specified categories <i>and are possibly in others</i>. #define DbMatchAny ((DbMatchModeType) 0x01) (OR) Match rows that are in <i>any</i> of the specified categories. #define DbMatchExact ((DbMatchModeType) 0x03) Match rows that are in <i>all</i> of the specified categories <i>and are in no others</i>.</pre>
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.

Character Encoding Types

Purpose Define the character encoding types for the encoding field of the [DBDatabaseInfo](#) structure.

Declared In SyncDb.h

```
#define charEncodingAscii ((CharEncodingType)1)
    Indicates ISO 646-1991 (ASCII).

#define charEncodingCP1252 ((CharEncodingType)7)
    Indicates the Windows variant of 8859-1, a Latin character
    encoding supported by Palm OS.

#define charEncodingCP932 ((CharEncodingType)8)
    Indicates the Windows variant of ShiftJIS, a Japanese
    character encoding supported by Palm OS.

#define charEncodingISO8859_1
    ((CharEncodingType)2)
    Indicates ISO 8859, Part 1, a Latin character encoding
    supported by Palm OS.

#define charEncodingPalmLatin
    ((CharEncodingType)3)
    Indicates the Palm OS version of CP1252, a Latin character
    encoding supported by Palm OS.

#define charEncodingPalmSJIS ((CharEncodingType)5)
    Indicates the Palm OS version of CP932, a Japanese character
    encoding supported by Palm OS.

#define charEncodingShiftJIS ((CharEncodingType)4)
    Indicates encoding for JIS 0208-1990 + 1-byte katakana, a
    Japanese character encoding supported by Palm OS.

#define charEncodingUnknown ((CharEncodingType)0)
    Indicates that the character encoding is unknown.

#define charEncodingUTF8 ((CharEncodingType)6)
    Indicates Unicode character encoding.
```

Comments These constants are defined by the Palm OS Text Manager and declared in `TextMgr.h`, which is provided in the SDK for Palm OS Cobalt.

Schema Sync Manager API

Character Encoding Types

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [DBDatabaseInfo](#)

Database Modification Reset Flag

Purpose	Indicates that the Sync Manager is to reset the modification state of a schema database when it is closed.
Declared In	SyncDb.h
Constants	#define SYNC_CLOSE_DB_OPT_SYNCED_ALL_CHANGES 0x20 Indicates that the database has been successfully synchronized and that the Sync Manager is to reset the modification state. When a conduit is finished synchronizing a schema database, it must call SyncDbCloseDatabase() with the <i>bOptFlags</i> parameter set to this value to reset the modification state; otherwise, sync atoms that were marked “modified” at the start of the HotSync operation remain so marked. A conduit may combine this value with one of the values in “ Database Closing Options ” on page 410.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncDbCloseDatabase() , “ Database Closing Options ” on page 410

Schema Sync Manager API

Database Information Retrieval Option for SyncDbReadOpenDatabaseInfo()

Database Information Retrieval Option for SyncDbReadOpenDatabaseInfo()

Purpose	Indicates that SyncDbReadOpenDatabaseInfo() retrieves extra information about a schema database.
Declared In	<code>SyncDb.h</code>
Constants	<pre>#define SYNC_DB_INFO_OPT_GET_SCHEMADB_FIELDS (0x10)</pre> <p>Indicates that database search operations are to retrieve encoding, table count, and display name fields in section 3 of the DBDatabaseInfo structure. This information is returned only for schema databases. You can use this value in the <i>bOptFlags</i> parameter of SyncDbReadOpenDatabaseInfo() only. You can also use the values defined in “Database Information Retrieval Options” on page 412.</p>
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncDbReadOpenDatabaseInfo() , DBDatabaseInfo , “ Database Information Retrieval Options ” on page 412

Database List Search Flags

Purpose	Indicate what types of databases to search for when reading a list of databases with SyncReadDatabaseList() .
Declared In	<code>SyncDb.h</code>
Constants	<pre>#define dbClassicDatabases ((UInt16) 0x0100) Indicates classic databases only. If you specify neither dbSchemaDatabases, dbExtendedDatabases, nor dbClassicDatabases, then the Sync Manager retrieves information about schema, extended, and classic databases. #define dbDatabasesAll ((UInt16) 0x0000) Indicates all databases. #define dbDatabasesInRAM ((UInt16) 0x0001) Indicates databases in RAM only. #define dbDatabasesInROM ((UInt16) 0x0002) Indicates databases in ROM only. If you specify neither dbDatabasesInRAM nor dbDatabasesInROM, then the Sync Manager retrieves information about databases in both RAM and ROM. #define dbExtendedDatabases ((UInt16) 0x0008) Indicates extended databases only. If you specify neither dbSchemaDatabases, dbExtendedDatabases, nor dbClassicDatabases, then the Sync Manager retrieves information about schema, extended, and classic databases. #define dbNonSecureDatabases ((UInt16) 0x0020) Indicates non-secure databases only. If you specify neither dbSecureDatabases nor dbNonSecureDatabases, then the Sync Manager does not filter databases based on whether they are secure. #define dbRecordDatabases ((UInt16) 0x0040) Indicates record databases only. This flag is mutually exclusive with dbResourceDatabases. #define dbResourceDatabases ((UInt16) 0x0080) Indicates resource databases only. This flag is mutually exclusive with dbRecordDatabases. If you specify neither dbRecordDatabases nor dbResourceDatabases, then the Sync Manager does not filter databases based on whether they contain records or resources.</pre>

Schema Sync Manager API

Database List Search Flags

```
#define dbSchemaDatabases ((UInt16) 0x0004)
    Indicates schema databases only.

#define dbSecureDatabases ((UInt16) 0x0010)
    Indicates secure schema databases only. Because only schema
    database can be secure, do not specify this flag along with
    dbClassicDatabases or dbExtendedDatabases. You
    may also explicitly specify dbSchemaDatabases, but it is
    implicit in dbSecureDatabases.
```

- Comments** Pass a valid combination of one or more of these values into the *searchFlags* parameter when you call [SyncReadDatabaseList\(\)](#).
For a summary of which characteristics are mutually exclusive, see [“Mutually Exclusive Database Characteristics”](#) on page 116 in *Introduction to Conduit Development*.
- Compatibility** Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.
- See Also** [SyncReadDatabaseList\(\)](#)

Database Open Modes

Purpose Defines the access modes in which a schema database can be opened.

Declared In SyncDb.h

Constants

```
#define dbModeReadOnly (dmModeReadOnly) // 0x0001
    Open with read-only access.
```

```
#define dbModeReadWrite (dmModeReadWrite) //
    0x0003
    Open with read/write access.
```

```
#define dbModeShowSecret (dmModeShowSecret) //
    0x0010
    Open with full access to rows marked private, which are
    rows with their dbRecAttrSecret bit set in
    DbRowData.attributes.
```

```
#define dmModeExclusive 0x0008
    While the database is open, do not let anyone else open it.
    This mode is not supported, so do not pass this value to
    SyncDbOpenDatabase().
```

```
#define dmModeWrite 0x0002
    Open the database with write-only access. This mode is not
    supported, so do not pass this value to
    SyncDbOpenDatabase(); use dbModeReadWrite instead.
```

Comments Palm OS does not support concurrent *write* access to the same database. Therefore specifying an open mode of dbModeReadWrite and a share mode of dbShareReadWrite is not supported and returns an error.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncDbOpenDatabase\(\)](#)

Schema Sync Manager API

Database Share Modes

Database Share Modes

Purpose	Define the shared access modes, which control how others can access a database that your application has opened.
Declared In	<code>SyncDb.h</code>
Constants	<pre>#define dbShareNone ((DbShareModeType) 0x0000) While the database is open, do not let anyone else open it. #define dbShareRead ((DbShareModeType) 0x0001) While the database is open, others can open it in read-only mode. #define dbShareReadWrite ((DbShareModeType) 0x0003) While the database is open, others can open it in read-only read-write mode.</pre>
Comments	Palm OS does not support concurrent <i>write</i> access to the same database. Therefore specifying an open mode of <code>dbModeReadWrite</code> and a share mode of <code>dbShareReadWrite</code> is not supported and returns an error.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	DbShareModeType , SyncDbOpenDatabase() , “ Database Open Modes ” on page 37

Maximum Name Lengths

Purpose	Define the maximum length of category, column, and table names for schema databases.
Declared In	SyncDb.h
Constants	<pre>#define catCategoryNameLength (32) The maximum number of ASCII bytes in a category name, including NULL terminator.</pre> <pre>#define dmColumnNameLength (32) The maximum number of ASCII bytes in a column name, including a NULL terminator.</pre> <pre>#define dmTableNameLength (32) The maximum number of ASCII bytes in a table name, including a NULL terminator.</pre>
Compatibility	Sync Manager version: 2.4 and later. Palm OS version: Palm OS Cobalt, version 6.0 or later.

Miscellaneous Schema Sync Manager Constants

Purpose	Define constants used with various schema Sync Manager functions.
Declared In	SyncDb.h
Constants	<pre>#define dbColumnPropertyUpperBound ((DbSchemaColumnProperty) 0x0A) Indicates the upper bound of the property IDs reserved for built-in properties; they range from 0x0 to this upper bound. See "DbSchemaColumnProperty" on page 22.</pre> <pre>#define DmVaultCreator ((UInt32) 'dmgr') The database creator ID reserved for the Data Manager vault database.</pre> <pre>#define DmVaultType ((UInt32) 'valt') The database creator ID reserved for all vault databases.</pre>
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.

Schema Sync Manager API

Row Attributes

Row Attributes

Purpose Define attributes of rows in a schema database when passed back in the `attributes` field in a [DbRowData](#) structure.

Declared In SyncDb.h

Constants

```
#define dbAllRecAttrs ( dbRecAttrDelete |
    dbRecAttrSecret | dbRecAttrArchive |
    dbRecAttrReadOnly)
```

A mask used to specify all row attributes.

```
#define dbRecAttrArchive 0x01
```

This value is not returned in the `DbRowData.attributes` field for a schema database row. Instead the `DbRowData.changeFlags` field is set to `dbRowArchived`.

```
#define dbRecAttrDelete dmRecAttrDelete
```

This value is not returned in the `DbRowData.attributes` field for a schema database row. Instead the `DbRowData.changeFlags` field is set to `dbRowDeleted`.

```
#define dbRecAttrReadOnly 0x02
```

The row is marked for read-only access. Note that neither the Data Manager nor the Sync Manager places any semantics on the read-only attribute. It is up to a handheld application and its conduit to enforce the read-only semantics. You can use the read-only attribute to implement certain row sharing scenarios on the handheld that allow a user to view a row but not to modify it.

```
#define dbRecAttrSecret dmRecAttrSecret
```

Indicates that the row has been marked as private and should be displayed only if the user wishes.

```
#define dbSysOnlyRecAttrs ( dbRecAttrDelete |
    dbRecAttrArchive )
```

A mask that specifies row attributes that only Palm OS can change.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [DbRowData](#)

Row Change Flags

Purpose	Define values for the <code>changeFlags</code> field of the DbRowData structure that indicate how or whether a row has been modified since the last HotSync operation.
Declared In	<code>SyncDb.h</code>
Constants	<pre>#define dbDataModified ((UInt16) 0x0001) Indicates that either a row's column data, the row header data, or both have been modified. #define dbDataNoChange ((UInt16) 0x0000) Indicates that <i>nothing</i> about a row—column values, category membership, and so on—has been modified. #define dbMembershipModified ((UInt16) 0x0002) Indicates that a row's category membership has been modified. #define dbRowArchived ((UInt16) 0x0008) Indicates that a row is marked for archiving. #define dbRowDeleted ((UInt16) 0x0004) Indicates that a row is marked for deletion. The row's column data no longer exists.</pre>
Comments	If the row has been modified, the value of the <code>DbRowInfo.changeFlags</code> is a combination of one or more of the following values: <code>dbDataModified</code> , <code>dbMembershipModified</code> , <code>dbRowDeleted</code> , or <code>dbRowArchived</code> .
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	DbRowData

Schema Sync Manager API

Row Data Sets

Row Data Sets

Purpose Indicate the defined sets of data or metadata that have been retrieved (or are to be retrieved) for a table row. These are the values of the [DbRowData](#).dataSetRetrieved field.

Declared In SyncDb.h

Constants

```
#define dbRowAllDataSet (0xffff)
    Complete row data. Includes all fields in a DbRowData
    structure.

#define dbRowCategoriesDataSet (0x0002)
    The numCategories and pCategoryIDList fields of the
    DbRowData structure. These values specify the row's
    category memberships.

#define dbRowColumnInfoDataSet (0x0004)
    The columnID, dataSize, and errCode fields of the
    DbColumnValue structures in the array that
    DbRowData.pColumnValues points to. The Sync Manager
    does not currently use this flag.

#define dbRowColumnValuesDataSet (0x0008)
    The DbRowData.numColumns value and all fields of the
    DbColumnValue structures in the array that
    DbRowData.pColumnValues points to.

#define dbRowEmptyDataSet (0x0000)
    No data. No fields in a DbRowData structure contain data.

#define dbRowInfoDataSet (0x0001)
    The row information fields of the DbRowData structure:
    errCode, rowID, attributes, pTableName, and
    changeFlags.
```

Comments	A valid combination of these flags in the DbRowData .dataSetRetrieved field indicate which other fields of this structure are to be retrieved by calls to SyncDbReadRow() , SyncDbReadRows() , SyncDbReadModifiedRows() , SyncDbReadRowInfo() , SyncDbReadRowInfoList() , SyncDbReadModifiedRowInfoList() , or SyncDbReadRowsByRowInfo() .
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	DbRowData , DbColumnValue , SyncDbReadRow() , SyncDbReadRows() , SyncDbReadModifiedRows() , SyncDbReadRowInfo() , SyncDbReadRowInfoList() , SyncDbReadModifiedRowInfoList() , SyncDbReadRowsByRowInfo()

Structure Sizes for Schema Databases

Purpose	Define the sizes of various structures used by schema Sync Manager functions.
Declared In	<code>SyncDb.h</code>
Constants	<pre>#define SIZEOF_DB_CATEGORY_DEFN (sizeof(DbCategoryDefn)) Defines the size of the DbCategoryDefn structure.</pre> <pre>#define SIZEOF_DB_COLUMN_DEFN (sizeof(DbColumnDefn)) Defines the size of the DbColumnDefn structure.</pre> <pre>#define SIZEOF_DB_COLUMN_REMOVE (sizeof(DbColumnRemove)) Defines the size of the DbColumnRemove structure.</pre> <pre>#define SIZEOF_DB_COLUMN_VALUE (sizeof(DbColumnValue)) Defines the size of the DbColumnValue structure.</pre> <pre>#define SIZEOF_DB_ROW_DATA (sizeof(DbRowData)) Defines the size of the DbRowData structure.</pre> <pre>#define SIZEOF_DB_ROW_RESULT (sizeof(DbRowResult)) Defines the size of the DbRowResult structure.</pre> <pre>#define SIZEOF_DB_SYNC_MODE (sizeof(DbSyncMode)) Defines the size of the DbSyncMode structure.</pre> <pre>#define SIZEOF_DB_TABLE_DEFN (sizeof(DbTableDefn)) Defines the size of the DbTableDefn structure.</pre>
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	DbCategoryDefn , DbColumnDefn , DbColumnRemove , DbColumnValue , DBDatabaseInfo , DbRowData , DbRowResult , DbSyncMode , DbTableDefn

Synchronization Flags

Purpose	Indicate whether a deleted sync atoms in a schema database has already been purged from the handheld.
Declared In	<code>SyncDb.h</code>
Constants	<pre>#define dbDeletesPurged ((UInt8) 0x01)</pre> <p>Indicates that deleted sync atoms have been purged from the handheld since the last HotSync operation with this desktop. The conduit must determine which sync atoms are purged from the handheld database and delete them from the desktop database.</p> <pre>#define dbNoFlag ((UInt8) 0x00)</pre> <p>Indicates that no additional information is available about the type of synchronization the conduit should perform.</p>
Comments	The <code>DbSyncType.flags</code> field takes one of these values. The “ Synchronization Types ” on page 46 provide information for a conduit to decide the fundamental type of synchronization to perform with a given schema database. All of this information is passed back when your conduit calls SyncDbGetSyncMode() .
<hr/> <p>NOTE: Conduits that synchronize schema databases must call SyncDbGetSyncMode() to determine they can perform a fast or slow sync with a given database.</p> <hr/>	
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	DbSyncType , SyncDbGetSyncMode() , “ Synchronization Types ” on page 46

Schema Sync Manager API

Synchronization Types

Synchronization Types

Purpose	Indicate whether a conduit needs to perform a fast, slow, or no synchronization on a given sync atoms in a schema database.
Declared In	<code>SyncDb.h</code>
Constants	<pre>#define dbFastSync ((UInt8) 0x01) Indicates that the sync atom has changed and that all the change flags are valid. #define dbNoChange ((UInt8) 0x00) Indicates that the sync atom has not changed. #define dbSlowSync ((UInt8) 0x02) Indicates that the change flags (or lack of change flags) for the sync atom cannot be trusted and only an object-by-object comparison can determine whether the sync atom has changed.</pre>
Comments	The DbSyncType .type field takes one of these values. The “ Synchronization Flags ” on page 45 further indicate whether a deleted sync atom has already been purged from the handheld. All of this information is passed back when your conduit calls SyncDbGetSyncMode() .
<p>NOTE: Conduits that synchronize schema databases must call SyncDbGetSyncMode() to determine they can perform a fast or slow sync with a given database.</p> <hr/>	
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	DbSyncType , SyncDbGetSyncMode() , “ Synchronization Flags ” on page 45

Table Column Attributes

Purpose	Define the attributes of a table column.
Declared In	<code>SyncDb.h</code>
Constants	<pre>#define dbAllSchemaColAttrs (dbSchemaColDynamic dbSchemaColNonSyncable dbSchemaColWritable)</pre> <p>The complete set of table column attributes.</p> <pre>#define dbSchemaColDynamic ((DbSchemaColumnAttrib)0x01)</pre> <p>Indicates that the column was added after the table was created.</p> <pre>#define dbSchemaColNonSyncable ((DbSchemaColumnAttrib)0x02)</pre> <p>Indicates that the column's data must not be synchronized with the desktop. Modifications made to a "non-syncable" column's data do not change the modification state for the row, and thus by themselves do not cause the Sync Manager to treat the row as modified.</p> <pre>#define dbSchemaColWritable ((DbSchemaColumnAttrib)0x04)</pre> <p>Indicates that the column's data is "always writable" even in a read-only row. Writable columns are relevant for read-only rows and are required for sharing. Note that the Sync Manager does not enforce this semantic; it is up to the handheld application or conduit to enforce it.</p>
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.

Schema Sync Manager API

Table Column Data Types

Table Column Data Types

Purpose	Define the permissible data types that columns can be defined as in a table.
Declared In	SyncDb.h
Constants	<pre>#define dbBlob ((DbSchemaColumnType) 0x11) An opaque array of bytes of variable length. This data type supports offset-based reads and writes.</pre> <pre>#define dbBoolean ((DbSchemaColumnType) 0x0B) A Boolean.</pre> <pre>#define dbChar ((DbSchemaColumnType) 0x0F) A character.</pre> <pre>#define dbDate ((DbSchemaColumnType) 0x0D) A date as defined by the DateType structure in the Palm OS DateType.h header file.</pre> <pre>#define dbDateTime ((DbSchemaColumnType) 0x0C) A date and time, not including seconds. This is defined by the DateTimeType structure in the Palm OS DateTime.h header file.</pre> <pre>#define dbDateTimeSecs ((DbSchemaColumnType) 0x12) A date and time, including seconds. This is defined by a uint32_t that stores time in the Unix epoch (seconds elapsed since January 1, 1970).</pre> <pre>#define dbDouble ((DbSchemaColumnType) 0x0A) A double.</pre> <pre>#define dbFloat ((DbSchemaColumnType) 0x09) A float.</pre> <pre>#define dbInt16 ((DbSchemaColumnType) 0x06) A signed 16-bit integer.</pre> <pre>#define dbInt32 ((DbSchemaColumnType) 0x07) A signed 32-bit integer.</pre> <pre>#define dbInt64 ((DbSchemaColumnType) 0x08) A signed 64-bit integer.</pre> <pre>#define dbInt8 ((DbSchemaColumnType) 0x05) A signed 8-bit integer.</pre>

```
#define dbStringVector ((DbSchemaColumnType) 0xC0)
    A string vector.

#define dbTime ((DbSchemaColumnType) 0x0E)
    A time as defined by the TimeType structure in the Palm OS
    DateTIme.h header file.

#define dbUInt16 ((DbSchemaColumnType) 0x02)
    An unsigned 16-bit integer.

#define dbUInt32 ((DbSchemaColumnType) 0x03)
    An unsigned 32-bit integer.

#define dbUInt64 ((DbSchemaColumnType) 0x04)
    An unsigned 64-bit integer.

#define dbUInt8 ((DbSchemaColumnType) 0x01)
    An unsigned 8-bit integer.

#define dbVarChar ((DbSchemaColumnType) 0x10)
    A character string of variable length. This data type supports
    offset-based reads and writes.

#define dbVector ((DbSchemaColumnType) 0x80)
    A vector. This data type supports offset-based reads and
    writes.
```

Comments These constants are used when adding columns to a table or getting table column definitions. For more on Palm OS data types, see the *Exploring Palm OS* documentation.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

Schema Sync Manager API

Schema Sync Manager Functions

Schema Sync Manager Functions

This section describes the following functions, which access only schema databases. See “[Common Sync Manager Functions and Macros](#)” on page 434 for functions that access all types of databases or utility functions and macros that do not access databases.

SyncDbAddCategory()	Adds a category to a schema database.
SyncDbAddColumns()	Adds one or more columns to a table.
SyncDbAddRowCategory()	Adds the specified set of categories to a row's category memberships.
SyncDbAddTable()	Adds a table to an existing schema database.
SyncDbCloseDatabase()	Closes a schema database and optionally updates the database's dates and modification state.
SyncDbCreateDatabase()	Creates a new schema database on the handheld, opens the database, and passes back a handle to it.
SyncDbCreateRow()	Creates a new row in a table.
SyncDbCreateRows()	Creates new rows in a table.
SyncDbDeleteAllRowsInTable()	Deletes all rows in a table.
SyncDbDeleteDatabase()	Deletes a schema database and all its rows.
SyncDbDeleteRow()	Deletes a row specified by ID.
SyncDbDeleteRows()	Deletes rows specified by a list of row IDs.
SyncDbDeleteRowsInCategory()	Deletes all rows that are members of the specified categories, depending on the given match mode criteria.

<u>SyncDbFindDatabase()</u>	Searches for a schema database by <i>name</i> and <i>creator ID</i> and retrieves information about the database, if it is found.
<u>SyncDbFindDatabaseByTypeCreator()</u>	Searches for a schema database by <i>creator ID</i> and <i>type</i> and retrieves information about the database, if it is found.
<u>SyncDbGetAllColumnValues()</u>	Retrieves all column values for a specified row.
<u>SyncDbGetCategoryCount()</u>	Retrieves the total number of categories in a schema database.
<u>SyncDbGetCategoryDefinitionList()</u>	Retrieves a list of the definitions of all the categories in a schema database.
<u>SyncDbGetChangeContext()</u>	Retrieves the change context for a schema database from the handheld.
<u>SyncDbGetColumnDefinitions()</u>	Retrieves the definitions of the specified table columns.
<u>SyncDbGetColumnIDs()</u>	Retrieves a list of all column IDs in a table.
<u>SyncDbGetColumnPropertyValue()</u>	Retrieves the property value of a table column.
<u>SyncDbGetColumnValue()</u>	Retrieves the specified bytes of a single column value for a row.
<u>SyncDbGetColumnValues()</u>	Retrieves one or more column value for a row.
<u>SyncDbGetModifiedCategoryDefinitionList()</u>	Retrieves a list of the definitions of all the modified categories in a schema database.
<u>SyncDbGetModifiedTableInfoList()</u>	Retrieves a list of names and column counts of all the modified tables in a schema database.

Schema Sync Manager API

Schema Sync Manager Functions

<u>SyncDbGetRowCategory()</u>	Retrieve the category membership for a row.
<u>SyncDbGetRowCountInTable()</u>	Retrieves the total number of rows in a table.
<u>SyncDbGetSyncMode()</u>	Determines the synchronization mode for a schema database.
<u>SyncDbGetTableCount()</u>	Retrieves the total number of tables in a schema database.
<u>SyncDbGetColumnInfo()</u>	Retrieves the number of columns in a table.
<u>SyncDbGetTableInfoList()</u>	Retrieves the number of columns in each of the specified tables.
<u>SyncDbGetTableSchema()</u>	Retrieves the schema for a table, including the definitions and built-in properties for all of the table's columns.
<u>SyncDbIsRowInCategory()</u>	Determines whether a row is a member of the specified categories, depending on the given match mode criteria.
<u>SyncDbModifyCategory()</u>	Modify a category in a schema database.
<u>SyncDbMoveRowsInCategory()</u>	Determines all the rows that are members of the specified categories, depending on the given match mode criteria, and moves them to a single target category.
<u>SyncDbOpenDatabase()</u>	Opens an existing schema database on the handheld.
<u>SyncDbPurgeAllRowsInTable()</u>	Removes all the rows that are marked as deleted or archived in a table.

<u>SyncDbReadModifiedRowInfoList()</u>	Reads information about all the rows that match the specified criteria and have been modified since the last HotSync operation.
<u>SyncDbReadModifiedRows()</u>	Reads all of a row's data—the attributes, category membership, and all column values—for all rows that match the specified criteria and have been modified since the last HotSync operation.
<u>SyncDbReadOpenDatabaseInfo()</u>	Retrieves information about an open schema database.
<u>SyncDbReadRow()</u>	Reads all of a row's data—the attributes, category membership, and all column values.
<u>SyncDbReadRowInfo()</u>	Reads information about a row.
<u>SyncDbReadRowInfoList()</u>	Reads information about all the rows that match the specified criteria.
<u>SyncDbReadRows()</u>	Reads all of a row's data—the attributes, category membership, and all column values—for all rows that match the specified criteria.
<u>SyncDbReadRowsByRowInfo()</u>	Reads all of a row's data—the attributes, category membership, and all column values—for an array of <u>DbRowData</u> structures.
<u>SyncDbReleaseStorage()</u>	Frees memory allocated by a call to a Sync Manager function that passes back a buffer.
<u>SyncDbRemoveCategory()</u>	Removes a category from a schema database.

Schema Sync Manager API

Schema Sync Manager Functions

SyncDbRemoveCategoryFromAllRows ()	Removes category membership in the specified categories from all rows in the database, depending on the match mode criteria.
SyncDbRemoveColumnProperty ()	Removes a single column property from a column in a database table.
SyncDbRemoveColumns ()	Removes one or more column definitions from a database schema and removes that column's data for all table rows described by that schema.
SyncDbRemoveRow ()	Removes a row specified by ID from a database and disposes of its data.
SyncDbRemoveRowCategory ()	Removes membership in the specified categories from a single row.
SyncDbRemoveRows ()	Removes rows specified by a list of row IDs from a database and disposes of their data.
SyncDbRemoveSecretRowsInTable ()	Removes all private rows from a table and disposes of their data.
SyncDbRemoveTable ()	Removes a table definition from a schema database.
SyncDbSetColumnPropertyValue ()	Sets a single property value for a table column.
SyncDbSetRowCategory ()	Sets category membership for a single database row.
SyncDbWriteColumnValue ()	Writes the specified bytes of a single column value for a row.
SyncDbWriteColumnValues ()	Writes one or more column values for a row.
SyncDbWriteRow ()	Writes all of a row's data—the attributes, category membership, and all column values.

Schema Sync Manager API

Schema Sync Manager Functions

[SyncDbWriteRows \(\)](#)

Writes all of the specified rows' data—the attributes, category membership, and all column values.

[SyncReadDatabaseList\(\)](#)

Retrieves information about a list of databases on the handheld.

Schema Sync Manager API

SyncDbAddCategory

SyncDbAddCategory Function

Purpose	Adds a category to a schema database.
Declared In	SyncDb.h
Prototype	HSError SyncDbAddCategory (HSByte <i>handle</i> , DbCategoryDefn * <i>pCategoryDefn</i>)
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>↔ <i>pCategoryDefn</i> A pointer to a DbCategoryDefn structure that specifies the name of the category to add. Upon return, it receives the category ID assigned by the handheld.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_MAX_CATEGORY_LIMIT Adding another category exceeds the maximum number of categories allowed.</p> <p>SYNCERR_NAME_ALREADY_EXISTS The category name specified in <i>*pCategoryDefn</i> already exists.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p>SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.</p> <p>SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.</p>

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments Category IDs are assigned by the Category Manager on the handheld.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [DbCategoryDefn](#), [CategoryID](#)

Schema Sync Manager API

SyncDbAddColumns

SyncDbAddColumns Function

Purpose	Adds one or more columns to a table.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbAddColumns (HSByte handle, const char *tableName, UInt32 numColumns, DbColumnDefn *pColumnDefnList)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>tableName</i> The name of the table to which to add columns.</p> <p>→ <i>numColumns</i> The number of columns in the <i>pColumnDefnList</i> array.</p> <p>↔ <i>pColumnDefnList</i> An array of <i>numColumns</i> DbColumnDefn structures that specifies the columns to add to the table. Upon return, it receives a success or error code in each column definition's <i>errCode</i> field.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_INVALID_TABLENAME The <i>tableName</i> parameter specifies an invalid table name or a valid table name that does not exist.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p>SYNCERR_ONE_OR_MORE_FAILED Addition of one or more column definitions failed. Check the <i>errCode</i> field of each DbColumnDefn in the <i>pColumnDefnList</i> array for details.</p>

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_TABLE_NAME_NOT_SPECIFIED

The *tableName* parameter specifies a null pointer or points to an empty character array.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

If this function returns SYNCERR_ONE_OR_MORE_FAILED, check the *errCode* field of each [*DbColumnDefn*](#) in the *pColumnDefnList* array to determine which columns were successfully added and which were not. For successfully added columns, the *errCode* is SYNCERR_NONE; otherwise it is set to one of these error codes:

SYNCERR_COLUMNID_ALREADY_EXISTS

The column ID specified in *columnID* already exists in the specified table.

SYNCERR_NAME_ALREADY_EXISTS

The column name specified in *columnName* already exists.

SYNCERR_INVALID_COLUMNTYPE

The column data type specified in *columnType* is not a defined type.

SYNCERR_INVALID_COLUMNSIZE

The column data size specified in *columnMaxSize* for a fixed-sized column is invalid.

SYNCERR_INVALID_COLUMNSPEC

One or more of the values specified in *columnAttributes* are invalid.

Each column that you add with this function has the *dbSchemaColDynamic* attribute set in its *DbColumnDefn.columnAttributes* field.

Schema Sync Manager API

SyncDbAddColumns

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [DbColumnDefn](#), [SyncDbRemoveColumns \(\)](#)

SyncDbAddRowCategory Function

Purpose	Adds the specified set of categories to a row's category memberships.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbAddRowCategory (HSByte handle, RowID rowID, UInt32 numCategories, const CategoryID *pCategoryIDList)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>rowID</i> The RowID value identifying the row to which to add category memberships.</p> <p>→ <i>numCategories</i> The number of categories in the <i>pCategoryIDList</i> array.</p> <p>→ <i>pCategoryIDList</i> An array of <i>numCategories</i> CategoryID values to add to the row's category membership list. This function adds the row to all the categories on this list.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_CATEGORYID A specified category ID doesn't correspond to a defined category.</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_NOT_FOUND The specified row is not present in the database.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p>

Schema Sync Manager API

SyncDbAddRowCategory

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

This function makes the specified row a member of one or more additional categories. It does not change the row’s existing category memberships.

The category IDs passed through the *pCategoryIDList* parameter must be valid category IDs. If any of the array values is not a valid category ID, then this function returns SYNCERR_INVALID_CATEGORYID.

If a given category ID value appears more than once in the *pCategoryIDList* array, the category membership is added only once. If the row is already a member of a category specified in the *pCategoryIDList* array, the array value is ignored and the row remains a member of that category.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[CategoryID](#), [SyncDbRemoveRowCategory\(\)](#),
[SyncDbSetRowCategory\(\)](#)

SyncDbAddTable Function

Purpose	Adds a table to an existing schema database.
Declared In	SyncDb.h
Prototype	HSError SyncDbAddTable (HSByte <i>handle</i> , const DbTableDefn * <i>pTableDefn</i>)
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>pTableDefn</i> A pointer to a DbTableDefn structure that specifies the table to add.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_COLUMNID_ALREADY_EXISTS At least one specified column ID is already defined in this table definition.</p> <p>SYNCERR_INVALID_COLUMNNAME At least one specified column name is invalid.</p> <p>SYNCERR_INVALID_COLUMNSIZE The column data size for a fixed-sized column is invalid.</p> <p>SYNCERR_INVALID_COLUMNSPEC At least one specified column attribute is not a valid column attribute.</p> <p>SYNCERR_INVALID_COLUMNTYPE At least one specified column data type is not a valid column data type.</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_INVALID_TABLENAME The *<i>pTableList</i>.name field specifies an invalid table name or a valid table name that does not exist.</p>

Schema Sync Manager API

SyncDbAddTable

SYNCERR_NAME_ALREADY_EXISTS

The specified table name already exists in this database. Or at least one column has the same name as another.

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_TABLE_NAME_NOT_SPECIFIED

The **pTableDefn.name* field specifies an empty character array.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

For a description of all Sync Manager error codes, see "[Schema Sync Manager Error Codes](#)" on page 207 and "[Common Sync Manager Error Codes](#)" on page 486.

Comments This function does not fill in the *errCode* fields of the [DbColumnDefn](#) structures in each table definition upon return.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [DbTableDefn](#), [DbColumnDefn](#), [SyncDbRemoveTable\(\)](#)

SyncDbCloseDatabase Function

Purpose	Closes a schema database and optionally updates the database's dates and modification state.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbCloseDatabase (HSByte handle, HSByte bOptFlags)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open in any open mode. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>bOptFlags</i> Options for updating the database upon closing. You can combine the constants defined in “Database Closing Options” on page 410 and “Database Modification Reset Flag” on page 33.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p>SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.</p> <p>SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.</p> <p>For a description of all Sync Manager error codes, see “Schema Sync Manager Error Codes” on page 207 and “Common Sync Manager Error Codes” on page 486.</p>
Comments	The Sync Manager allows multiple schema databases to be open at once, unlike classic databases. Therefore you need to close schema

Schema Sync Manager API

SyncDbCloseDatabase

databases only before deleting them or before exiting your conduit. However, because of the overhead incurred by an open database, you should close a schema database as soon as you no longer need it open.

NOTE: A conduit must pass in the `SYNC_CLOSE_DB_OPT_SYNCED_ALL_CHANGES` value via `bOptFlags` to indicate that the schema database has been successfully synchronized and that the Sync Manager is to reset the modification state. For a schema database, calling this function with this flag is equivalent to calling the “reset sync flags” functions for non-schema databases.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncDbOpenDatabase\(\)](#), [SyncDbCreateDatabase\(\)](#)

SyncDbCreateDatabase Function

Purpose Creates a new schema database on the handheld, opens the database, and passes back a handle to it.

Declared In SyncDb.h

Prototype

```
HSError SyncDbCreateDatabase (const char *pName,
                             UInt32 creator, UInt32 type,
                             UInt16 attributes, UInt16 version,
                             UInt32 numTables,
                             const DbTableDefn *pTableList,
                             HSByte *pHandle)
```

Parameters

- *pName*
The [database name](#).
- *creator*
The database [creator ID](#).
- *type*
The [database type](#).
- *attributes*
A combination of one or more nonexclusive attributes of this schema database. Specify values described in “[Database Attributes](#)” on page 407.
- *version*
The version of the database. This value is application-specific and defaults to 0 if not specified.
- *numTables*
The number of tables in *pTableList*. This parameter can be zero, which creates a new database with no tables defined.
- *pTableList*
An array of *numTables* [DbTableDefn](#) structures. Each element defines the schema for a newly-created database table.
- ← *pHandle*
Passes back a handle to the newly-created schema database. Pass this handle to all schema Sync Manager functions that have an input *handle* parameter.

Returns

If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following error code values:

Schema Sync Manager API

SyncDbCreateDatabase

SYNCERR_BAD_ARG

One or more of the parameters passed in are invalid—for example, *pName* is NULL or an empty string.

SYNCERR_COLUMNID_ALREADY_EXISTS

At least one specified column ID is defined more than once in a given table definition.

SYNCERR_FILE_ALREADY_EXIST

Another database with this name already exists.

SYNCERR_INVALID_COLUMNNAME

At least one specified column name is invalid.

SYNCERR_INVALID_COLUMNSIZE

The column data size for a fixed-sized column is invalid.

SYNCERR_INVALID_COLUMNSPEC

At least one specified column attribute is not a valid column attribute.

SYNCERR_INVALID_COLUMNTYPE

At least one specified column data type is not a valid column data type.

SYNCERR_INVALID_TABLENAME

The name field in a *pTableList* entry specifies an invalid table name or a valid table name that does not exist.

SYNCERR_NAME_ALREADY_EXISTS

One of the specified table names occurs in more than one *pTableList* entry.

SYNCERR_TABLE_NAME_NOT_SPECIFIED

The name field in a *pTableList* entry points to an empty character array.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

After this function creates the database, it opens the database for read-write access in exclusive sharing mode, with the default sort index and with private (secret) records shown.

The Sync Manager allows multiple schema databases to be open at the same time. Though a conduit must close its open databases before exiting; otherwise, other conduits will not be able to open the same databases for writing later in the same HotSync operation.

NOTE: `SyncDbCreateDatabase()` does not fill in the `errCode` fields of the [DbColumnDefn](#) structures in each table definition upon return.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[DbTableDefn](#), [DbColumnDefn](#), “[Database Attributes](#)” on page 407, [SyncDbCloseDatabase\(\)](#)

Schema Sync Manager API

SyncDbCreateRow

SyncDbCreateRow Function

Purpose	Creates a new row in a table.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbCreateRow (HSByte handle, const char *tableName, UInt16 attributes, UInt32 numCategories, const CategoryID *pCategoryIDList, UInt32 numColumns, DbColumnValue *pColumnValues, RowID *pRowID)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>tableName</i> The name of the table to which to add a row.</p> <p>→ <i>attributes</i> A combination of one or more nonexclusive attributes of this row. Specify values described in “Row Attributes” on page 40.</p> <p>→ <i>numCategories</i> The number of categories in the <i>pCategoryIDList</i> array.</p> <p>→ <i>pCategoryIDList</i> An array of <i>numCategories</i> CategoryID values to add to the row’s category membership list. This function adds the row to all the categories on this list.</p> <p>→ <i>numColumns</i> The number of column values in the <i>pColumnValues</i> array.</p> <p>→ <i>pColumnValues</i> An array of <i>numColumns</i> DbColumnValue structures that specifies the column values of the new row. Upon return, it receives a success or error code in each column value’s <i>errCode</i> field.</p> <p>↔ <i>pRowID</i> Upon return, a pointer to the RowID value that the handheld assigned to the newly created row. However, in the case of a restore operation (desktop overwrites handheld), a conduit can specify the row ID.</p>

Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following error code values: SYNCERR_BAD_ARG One or more of the parameters passed in are invalid. SYNCERR_INVALID_CATEGORYID A category ID doesn't correspond to a defined category. SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value. SYNCERR_INVALID_TABLENAME The <i>tableName</i> parameter specifies an invalid table name or a valid table name that does not exist. SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases. SYNCERR_ONE_OR_MORE_FAILED Creating one or more columns failed. Check the <i>errCode</i> field of each DbColumnValue in the <i>pColumnValues</i> array for details. SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld. SYNCERR_TABLE_NAME_NOT_SPECIFIED The <i>tableName</i> parameter specifies a null pointer or points to an empty character array. SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below. For a description of all Sync Manager error codes, see " Schema Sync Manager Error Codes " on page 207 and " Common Sync Manager Error Codes " on page 486.
Comments	If <i>numColumns</i> is zero or <i>pColumnValues</i> is NULL, a row without column values is created—which may subsequently be written into using either SyncDbWriteColumnValue() or SyncDbWriteColumnValues() . Normally, the Data Manager on the handheld assigns row IDs, which this function passes it back when a conduit creates a row. An

Schema Sync Manager API

SyncDbCreateRow

exception is a restore operation (desktop overwrites handheld): a conduit first either deletes the database and recreates it, or purges all deleted rows and creates new rows. To create new rows in this case, a conduit can call `SyncDbCreateRow()` or [SyncDbCreateRows\(\)](#) and pass in row ID values.

If this function returns `SYNCERR_ONE_OR_MORE_FAILED`, check the `errCode` field of each [`DbColumnValue`](#) structure in the `pColumnValues` array to determine which column values were successfully added and which were not. For successfully added column values, the `errCode` is `SYNCERR_NONE`; otherwise it is set to one of these error codes:

`SYNCERR_INVALID_COLUMNID`

The column ID specified in `columnID` is invalid.

`SYNCERR_INVALID_COLUMNSIZE`

The column data size specified in `dataSize` is invalid.

`SYNCERR_OPERATION_ABORTED`

Writing column values has aborted because of one or more invalid input parameters.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[`DbColumnValue`](#), [SyncDbCreateRows\(\)](#), [SyncDbDeleteRow\(\)](#), [SyncDbRemoveRow\(\)](#)

SyncDbCreateRows Function

Purpose	Creates new rows in a table.
Declared In	SyncDb.h
Prototype	HSError SyncDbCreateRows (HSByte <i>handle</i> , UInt32 <i>numRows</i> , DbRowData * <i>prowData</i>)
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>numRows</i> The number of rows in the <i>prowData</i> array.</p> <p>↔ <i>prowData</i> An array of <i>numRows</i> DbRowData structures that specifies the data and metadata for the new rows, including the table in which to create these rows. Upon return, it receives a success or error code in each row's <i>errCode</i> field.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p>SYNCERR_ONE_OR_MORE_FAILED Creating one or more rows failed. Check the <i>errCode</i> field of each DbRowData in the <i>prowData</i> array for details.</p> <p>SYNCERR_STREAM_ERR An error occurred while streaming the data to the handheld.</p>

Schema Sync Manager API

SyncDbCreateRows

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

In the *pRowData* array, you must initialize all fields in each [*DbRowData*](#) structure. In the *pCategoryIDList* field, you cannot specify either *catIDUnfiled* or *catIDAll*. Always initialize the *errCode* field to *SYNCERR_NONE*.

If row creation fails for one row, this function continues to create the next row. If one or more rows cannot be created, this function returns *SYNCERR_ONE_OR_MORE_FAILED*. The *errCode* field for the failed rows is set to the appropriate error value. For the rows that can be successfully created, the Schema Database Manager on the handheld sets the *rowID* field to the corresponding row ID value. Note that only during a restore operation

([*CSyncProperties*](#)::*m_SyncType* is *ePCToHH*) can a conduit specify the row ID for each new row. In this case, a conduit can specify only the row ID stored on the desktop that was originally assigned by the handheld; it cannot use its own scheme for assigning row IDs

If this function returns *SYNCERR_ONE_OR_MORE_FAILED*, check the *errCode* field of each [*DbRowData*](#) structure in the *pRowData* array to determine which rows were successfully added and which were not. For successfully added rows, the *errCode* is *SYNCERR_NONE*; otherwise it is set to one of these error codes:

SYNCERR_INVALID_TABLENAME

The table name specified in **pTableName* is invalid or does not exist.

SYNCERR_INVALID_CATEGORYID

The allowed number of categories has been exceeded, or a category ID doesn't correspond to a defined category.

SYNCERR_INVALID_ID

The row ID specified in *rowID* is invalid—for example, because a row already exists with the same row ID. This error can be returned only during a restore operation, which typically happens only after the handheld is hard-reset.

SYNCERR_TABLE_NAME_NOT_SPECIFIED

The pTableName specifies a null pointer or points to an empty character array.

SYNCERR_ONE_OR_MORE_FAILED

Creating one or more columns failed. Check the `errCode` field of each [DbColumnValue](#) in the `pColumnValues` array for details.

If the `errCode` field of a [DbRowData](#) structure in the `pRowData` array returns `SYNCERR_ONE_OR_MORE_FAILED`, check the `errCode` field of each [DbColumnValue](#) structure in the `pColumnValues` array to determine which column values were successfully added and which were not. For successfully added column values, the `errCode` is `SYNCERR_NONE`; otherwise it is set to one of these error codes:

SYNCERR_INVALID_COLUMNID

The column ID specified in `columnID` is invalid.

SYNCERR_INVALID_COLUMNSIZE

The column data size specified in `dataSize` is invalid.

SYNCERR_OPERATION_ABORTED

Writing column values has aborted because of one or more invalid input parameters.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[DbRowData](#), [DbColumnValue](#), [SyncDbCreateRow\(\)](#),
[SyncDbDeleteRows\(\)](#)

Schema Sync Manager API

SyncDbDeleteAllRowsInTable

SyncDbDeleteAllRowsInTable Function

Purpose	Deletes all rows in a table.
Declared In	SyncDb.h
Prototype	HSError SyncDbDeleteAllRowsInTable (HSByte <i>handle</i> , const char * <i>tableName</i>)
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading and writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>tableName</i> The name of the table from which to delete all rows.</p>
Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following error code values: SYNCERR_BAD_ARG One or more of the parameters passed in are invalid. SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value. SYNCERR_INVALID_TABLENAME The <i>tableName</i> parameter specifies an invalid table name or a valid table name that does not exist. SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases. SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld. SYNCERR_TABLE_NAME_NOT_SPECIFIED The <i>tableName</i> parameter specifies a null pointer or points to an empty character array. SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below. For a description of all Sync Manager error codes, see “ Schema Sync Manager Error Codes ” on page 207 and “ Common Sync Manager Error Codes ” on page 486.

Comments	For the specified database and table, this function immediately disposes of all the rows' data, but it retains the rows' entries in the database header.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncDbDeleteRow() , SyncDbDeleteRows() , SyncDbDeleteRowsInCategory() , SyncDbPurgeAllRowsInTable() , SyncDbCreateRows()

Schema Sync Manager API

SyncDbDeleteDatabase

SyncDbDeleteDatabase Function

Purpose	Deletes a schema database and all its rows.
Declared In	SyncDb.h
Prototype	HSError SyncDbDeleteDatabase (const char * <i>pName</i> , UInt32 <i>creator</i>)
Parameters	<p>→ <i>pName</i> The database name.</p> <p>→ <i>creator</i> The database creator ID.</p>
Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following error code values: SYNCERR_ACCESS_DENIED Access is denied by the Authorization Manager on the handheld or the database cannot be unencrypted. SYNCERR_BAD_ARG One or more of the parameters passed in are invalid. SYNCERR_FILE_ALREADY_OPEN The specified database must be closed before you call this function. SYNCERR_NOT_FOUND The specified database is not present on the handheld. SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below. For a description of all Sync Manager error codes, see “ Schema Sync Manager Error Codes ” on page 207 and “ Common Sync Manager Error Codes ” on page 486.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncDbCloseDatabase() , SyncDbCreateDatabase()

SyncDbDeleteRow Function

Purpose	Deletes a row specified by ID.
Declared In	SyncDb.h
Prototype	HSError SyncDbDeleteRow (HSByte <i>handle</i> , RowID <i>rowID</i>)
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading and writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>rowID</i> The RowID value identifying the row to delete.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p>SYNCERR_NOT_FOUND The specified row is not in the database.</p> <p>SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.</p> <p>For a description of all Sync Manager error codes, see “Schema Sync Manager Error Codes” on page 207 and “Common Sync Manager Error Codes” on page 486.</p>

Schema Sync Manager API

SyncDbDeleteRow

Comments	This function immediately disposes of the row's data, but it retains the row's entry in the database header.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncDbDeleteAllRowsInTable() , SyncDbDeleteRows() , SyncDbDeleteRowsInCategory() , SyncDbPurgeAllRowsInTable() , SyncDbCreateRows()

SyncDbDeleteRows Function

Purpose	Deletes rows specified by a list of row IDs.
Declared In	<code>SyncDb.h</code>
Prototype	<code>HSError SyncDbDeleteRows (HSByte handle, UInt32 numRows, DbRowResult *pRowDelete)</code>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading and writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>numRows</i> The number of rows in the <i>pRowData</i> array.</p> <p>↔ <i>pRowDelete</i> An array of <i>numRows</i> DbRowResult structures that specifies the row ID of each row to delete. Upon return, it receives a success or error code in each entry's <i>errCode</i> field.</p>
Returns	<p>If successful, returns <code>SYNCERR_NONE</code>.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p><code>SYNCERR_BAD_ARG</code> One or more of the parameters passed in are invalid.</p> <p><code>SYNCERR_INVALID_FILE_HANDLE</code> The <i>handle</i> parameter is 0, which is an invalid value.</p> <p><code>SYNCERR_NOT_SCHEMADB</code> The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p><code>SYNCERR_ONE_OR_MORE_FAILED</code> Deleting one or more rows failed. Check the <i>errCode</i> field of each DbRowResult in the <i>pRowDelete</i> array for details.</p> <p><code>SYNCERR_REMOTE_BAD_ARG</code> One or more invalid parameters has been passed to the handheld.</p> <p><code>SYNCERR_UNKNOWN_REQUEST</code> The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.</p>

Schema Sync Manager API

SyncDbDeleteRows

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments	This function immediately disposes of the rows' data, but it retains the rows' entries in the database header. In the <i>pRowDelete</i> array, you must initialize all fields in each <i>DbRowResult</i> structure. Always initialize the <i>errCode</i> field to <i>SYNCERR_NONE</i> . If this function returns <i>SYNCERR_ONE_OR_MORE_FAILED</i> , check the <i>errCode</i> field of each <i>DbRowResult</i> structure in the <i>pRowDelete</i> array to determine which rows were successfully deleted and which were not. For successfully deleted rows, the <i>errCode</i> is <i>SYNCERR_NONE</i> ; otherwise it is set to one of these error codes: SYNCERR_NOT_FOUND The specified row is not in the database. SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncDbDeleteAllRowsInTable() , SyncDbDeleteRow() , SyncDbDeleteRowsInCategory() , SyncDbPurgeAllRowsInTable() , SyncDbCreateRows()

SyncDbDeleteRowsInCategory Function

Purpose Deletes all rows that are members of the specified categories, depending on the given match mode criteria.

Declared In SyncDb.h

Prototype

```
HSError SyncDbDeleteRowsInCategory (HSByte handle,
                                     UInt32 numCategories,
                                     const CategoryID *pCategoryIDList,
                                     DbMatchModeType matchMode)
```

Parameters

- *handle*
A handle to a schema database that is open for reading and writing. This handle is returned by [SyncDbOpenDatabase\(\)](#) or [SyncDbCreateDatabase\(\)](#).
- *numCategories*
The number of categories in the *pCategoryIDList* array.
- *pCategoryIDList*
An array of *numCategories* [CategoryID](#) values.
- *matchMode*
One of the following [DbMatchModeType](#) values that specifies how the categories in the *pCategoryIDList* array must match a given row's category membership for this function to delete the row:
 - DbMatchAny
(OR) Delete rows that are in *any* of the categories specified in the *pCategoryIDList* array.
 - DbMatchAll
(AND) Delete rows that are in *all* of the categories specified in the *pCategoryIDList* array *and are possibly in others*.
 - DbMatchExact
Delete rows that are in *all* of the categories specified in the *pCategoryIDList* array *and are in no others*.

Schema Sync Manager API

SyncDbDeleteRowsInCategory

Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following error code values: SYNCERR_BAD_ARG One or more of the parameters passed in are invalid. SYNCERR_INVALID_CATEGORYID The allowed number of categories has been exceeded, or a category ID doesn't correspond to a defined category. SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value. SYNCERR_INVALID_MATCH_MODE The <i>matchMode</i> parameter specifies an invalid match mode. Specify only values described in " Category Match Modes " on page 30. SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases. SYNCERR_REMOTE_BAD_ARG One or more invalid parameters have been passed to the handheld. SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below. For a description of all Sync Manager error codes, see " Schema Sync Manager Error Codes " on page 207 and " Common Sync Manager Error Codes " on page 486.
Comments	This function immediately disposes of the matching rows' data, but it retains the rows' entries in the database header.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	"Category Match Modes" on page 30

SyncDbFindDatabase Function

Purpose Searches for a schema database by *name* and *creator ID* and retrieves information about the database, if it is found.

Declared In SyncDb.h

Prototype

```
HSError SyncDbFindDatabase (const char *pName,
                             UInt32 creator, HSByte bOptFlags,
                             DBDatabaseInfo *pDatabaseInfo)
```

Parameters

→ *pName*
The [database name](#).

→ *creator*
The database [creator ID](#).

→ *bOptFlags*
Specifies additional information to retrieve about the matching database. Pass in a combination of one or more of the following values:

SYNC_DB_INFO_OPT_GET_ATTRIBUTES
Retrieve database attributes.

SYNC_DB_INFO_OPT_GET_SIZE
Retrieve count and data size information.

These values are defined in “[Database Information Retrieval Options](#)” on page 412.

← *pDatabaseInfo*
A pointer to a [DBDatabaseInfo](#) structure that receives information about the matching database. Do not pass in NULL.

Schema Sync Manager API

SyncDbFindDatabase

Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following error code values: SYNCERR_BAD_ARG One or more of the parameters passed in are invalid. SYNCERR_NOT_FOUND The specified schema database is not on the handheld. SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld. SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below. For a description of all Sync Manager error codes, see " Schema Sync Manager Error Codes " on page 207 and " Common Sync Manager Error Codes " on page 486.
Comments	This function searches for a schema database only; it does not include extended or classic databases in its search.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	DBDatabaseInfo , SyncDbFindDatabaseByTypeCreator()

SyncDbFindDatabaseByTypeCreator Function

Purpose Searches for a schema database by *creator ID* and *type* and retrieves information about the database, if it is found.

Declared In SyncDb.h

Prototype

```
HSError SyncDbFindDatabaseByTypeCreator
    (UInt32 creator, UInt32 type,
     HSByte searchFlag, HSByte bOptFlags,
     DBDatabaseInfo *pDatabaseInfo)
```

Parameters

→ *creator*
The database [creator ID](#).

→ *type*
The [database type](#).

→ *searchFlag*
The search options for the find operation. This parameter determines whether this call searches for only the latest version of the database and whether the call is the start of a new search. Specify one or more of the values defined in “[Database Search Options](#)” on page 413 (SYNC_DB_SRCH_OPT_...). If you pass in zero, then this function continues the search for the next matching database, regardless of version.

→ *bOptFlags*
Specifies additional information to retrieve about the matching database. Pass in a combination of one or more of the following values:

SYNC_DB_INFO_OPT_GET_ATTRIBUTES
Retrieve database attributes.

SYNC_DB_INFO_OPT_GET_SIZE
Retrieve count and data size information.

These values are defined in “[Database Information Retrieval Options](#)” on page 412.

← *pDatabaseInfo*
A pointer to a [DBDatabaseInfo](#) structure that receives information about the matching database. Do not pass in NULL.

Schema Sync Manager API

SyncDbFindDatabaseByTypeCreator

Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following error code values: SYNCERR_BAD_ARG One or more of the parameters passed in are invalid. SYNCERR_NOT_FOUND The specified schema database is not on the handheld. SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld. SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below. For a description of all Sync Manager error codes, see " Schema Sync Manager Error Codes " on page 207 and " Common Sync Manager Error Codes " on page 486.
Comments	This function searches for a schema database only; it does not include extended or classic databases in its search.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	DBDatabaseInfo , SyncDbFindDatabase()

SyncDbGetAllColumnValues Function

Purpose	Retrieves all column values for a specified row.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbGetAllColumnValues (HSByte handle, RowID rowID, UInt32 *pNumColumnValues, DbColumnValue *pColumnValues)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>rowID</i> The RowID value identifying the row for which to retrieve column values.</p> <p>↔ <i>pNumColumnValues</i> Specifies the number of elements that the caller allocated in the <i>pColumnValues</i> array. The caller can pass in an address to a zero value, but it must not pass in NULL. Upon return, receives the number of column values that this function retrieves, if the number of elements in the <i>pColumnValues</i> array is sufficient; or the number of columns in the row on the handheld, if the number of elements in the <i>pColumnValues</i> array is insufficient.</p> <p>← <i>pColumnValues</i> An array of <i>*pNumColumnValues</i> structures of type DbColumnValue that receives the row's column values and error codes that indicate whether each column value is retrieved successfully. The caller can specify NULL, but only if <i>*pNumColumnValues</i> is 0.</p>

Schema Sync Manager API

SyncDbGetAllColumnValues

Returns If successful, returns one of the following values:

SYNCERR_MORE

The *pColumnValues* array is not large enough. Allocate this array at least to the size passed back in *pNumColumnValues* and call this function again to retrieve all column values. This error is also returned if any of the *columnData* values have not been retrieved in any [DbColumnValue](#) structure in the *pColumnValues* array. In this case, allocate enough memory and assign it to the *columnData* field.

SYNCERR_NONE

The *pColumnValues* array contains all the column values.

If unsuccessful, returns one of the following error code values:

SYNCERR_BAD_ARG

One or more of the parameters passed in are invalid.

SYNCERR_INVALID_FILE_HANDLE

The *handle* parameter is 0, which is an invalid value.

SYNCERR_NOT_FOUND

The specified row is not present in the database.

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

This function requires that the caller allocate memory for the *pColumnValues* array of [DbColumnValue](#) structures, as necessary.

If you do not know the number of columns in the row, you can call this function to retrieve the column count in **pNumColumnValues*: pass in NULL in *pColumnValues* and 0 in **pNumColumnValues*.

This function passes back in **pNumColumnValues* the number of columns actually in this row and returns SYNCERR_MORE..

Now that you know the number of columns in the row (or knew it already), call this function again with a *pColumnValues* array and **pNumColumnValues* value at least the size passed back from the first call. Upon return, this function passes back in *pNumColumnValues* the number of columns actually in this row. If in each [DbColumnValue](#) structure in the array you set the *columnData* field to NULL, you must also set the *dataSize* field to 0. However, if you set *dataSize* to a nonzero value, then *columnData* must not be NULL. In either case, if *dataSize* is less than the actual size of the data available for that column value, then this function passes back in the actual *dataSize* value and sets the *errCode* field to SYNCERR_MORE. From this information, you can allocate sufficient memory for the *pColumnValues* array and each *columnData* field, fill in the *dataSize* fields appropriately, initialize all other fields, and call this function a third time, which fills in the array with column values.

The first time you call this function, the Sync Manager retrieves the column values from the handheld and caches them on the desktop. The cache is preserved until you call a different Sync Manager function. If the call is not for the cached data, the cache is emptied. Therefore multiple calls to this function do not incur the performance penalty of retrieving data from the handheld each time over a possibly slow connection.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [DbColumnValue](#), [SyncDbGetColumnValue \(\)](#),
[SyncDbGetColumnValues \(\)](#)

Schema Sync Manager API

SyncDbGetCategoryCount

SyncDbGetCategoryCount Function

Purpose	Retrieves the total number of categories in a schema database.
Declared In	SyncDb.h
Prototype	HSError SyncDbGetCategoryCount (HSByte <i>handle</i> , UInt32 * <i>pNumCategories</i>)
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>← <i>pNumCategories</i> A pointer to the category count.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p>SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.</p> <p>SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.</p> <p>For a description of all Sync Manager error codes, see “Schema Sync Manager Error Codes” on page 207 and “Common Sync Manager Error Codes” on page 486.</p>
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncDbGetCategoryDefinitionList()

SyncDbGetCategoryDefinitionList Function

Purpose Retrieves a list of the definitions of all the categories in a schema database.

Declared In SyncDb.h

Prototype

```
HSError SyncDbGetCategoryDefinitionList
    (HSByte handle, UInt32 *pNumElements,
     DbCategoryDefn *pCategoryDefnList)
```

Parameters

→ *handle*
A handle to a schema database that is open for reading. This handle is returned by [SyncDbOpenDatabase\(\)](#) or [SyncDbCreateDatabase\(\)](#).

↔ *pNumElements*

Specifies the number of elements that the caller allocates in the *pCategoryDefnList* array. The caller can pass in an address to a zero value, but it must not pass in NULL. Upon return, receives the number of category definitions that this function retrieves, if the number of elements in the *pCategoryDefnList* array is sufficient; or the number of category definitions in the database on the handheld, if the number of elements in the *pCategoryDefnList* array is insufficient.

← *pCategoryDefnList*

An array of **pNumElements* structures of type [DbCategoryDefn](#) that receives the category ID and name of each category. The caller can specify NULL, but only if **pNumElements* is 0.

Schema Sync Manager API

SyncDbGetCategoryDefinitionList

Returns If successful, returns one of the following values:

SYNCERR_MORE

The *pCategoryDefnList* array is not large enough.
Allocate this array to the size passed back in *pNumElements*
and call this function again to retrieve all category
definitions.

SYNCERR_NONE

The *pCategoryDefnList* array contains all the category
definitions.

If unsuccessful, returns one of the following error code values:

SYNCERR_BAD_ARG

One or more of the parameters passed in are invalid.

SYNCERR_INVALID_FILE_HANDLE

The *handle* parameter is 0, which is an invalid value.

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This
function operates only on schema databases.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the
handheld.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this
function does not support. See "Compatibility" below.

For a description of all Sync Manager error codes, see "[Schema Sync Manager Error Codes](#)" on page 207 and "[Common Sync Manager Error Codes](#)" on page 486.

Comments

This function requires that the caller allocate memory for the
pCategoryDefnList array of [*DbCategoryDefn*](#) structures, as
necessary.

If you do not know the number of categories in the database, you
can call this function to retrieve the category count in

**pNumElements*: pass in NULL in *pCategoryDefnList* and 0 in
**pNumElements*. This function passes back in **pNumElements*
the number of categories actually in the database and returns
SYNCERR_MORE.. Alternatively, you can call

[SyncDbGetCategoryCount \(\)](#) to retrieve only the number of categories.

Now that you know the number of categories in the database (or knew it already), call this function again with a *pCategoryDefnList* array and **pNumElements* value equal to the size passed back from the first call.

The first time you call this function the Sync Manager retrieves the category definitions from the handheld and caches them on the desktop. The cache is preserved until you call the next Sync Manager function. If the call is not for the cached data, the cache is emptied. Therefore multiple calls to this function do not incur the performance penalty of retrieving data from the handheld each time over a possibly slow connection.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [DbCategoryDefn](#), [SyncDbGetCategoryCount \(\)](#)

Schema Sync Manager API

SyncDbGetChangeContext

SyncDbGetChangeContext Function

Purpose Retrieves the change context for a schema database from the handheld.

Declared In SyncDb.h

Prototype HSError SyncDbGetChangeContext (const char **pName*,
 UInt32 *creator*, UInt32 *type*,
 UInt32 **pChangeContextLength*,
 HSByte **pChangeContext*)

Parameters

→ <i>pName</i>	The database name .
→ <i>creator</i>	The database creator ID .
→ <i>type</i>	The database type .
↔ <i>pChangeContextLength</i>	Specifies the number of bytes that the caller allocates in the <i>pChangeContext</i> array. The caller can pass in an address to a zero value, but it must not pass in NULL. Upon return, receives the length, in bytes, of the change context that this function retrieves, if the size of the <i>pChangeContext</i> array is sufficient; or the actual length of the change context, if the size of the <i>pChangeContext</i> buffer is insufficient.
← <i>pChangeContext</i>	An array of bytes that receives the change context. The caller can specify NULL, but only if * <i>pChangeContextLength</i> is 0.

Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following error code values: SYNCERR_BAD_ARG One or more of the parameters passed in are invalid. SYNCERR_LOCAL_BUFF_TOO_SMALL The byte array specified by <i>pChangeContext</i> is too small. SYNCERR_NOT_FOUND The change context is not available for this database, because this database was never synchronized successfully. SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.
	For a description of all Sync Manager error codes, see " Schema Sync Manager Error Codes " on page 207 and " Common Sync Manager Error Codes " on page 486.

Comments	The Sync Manager stores the change context per user, conduit, and schema database and retrieves it automatically to determine the type of synchronization operation. Therefore most conduits do not need to store the change context themselves, but instead can rely on the Sync Manager to handle it. However, in some special cases a conduit might need to store the change context itself. For example, if the data source that the conduit synchronizes with resides in a central location, and the user synchronizes from multiple desktops with the same central data source using the same conduit, the conduit can store this change context in the same central location. Then during subsequent HotSync operations from different desktops, the conduit can retrieve the change context and pass it to the Sync Manager via SyncDbGetSyncMode() to determine the synchronization mode that the conduit should use. After comparing the change contexts, the Sync Manager might indicate a fast sync is possible when it would not have been possible otherwise.
-----------------	---

NOTE: A conduit must call this function *after* closing the database, but only if it needs to store a database's change context itself. Most conduits do not.

Schema Sync Manager API

SyncDbGetChangeContext

This function requires that the caller allocate memory for the *pChangeContext* byte array, as necessary.

To retrieve the size of the change context, set *pChangeContextLength* to point to a zero value and *pChangeContext* to NULL. Specify the name, creator ID, and type of the closed schema database and call this function. Then the Sync Manager retrieves the change context from the handheld, caches it, sets **pChangeContextLength* to the actual size of the change context, and returns SYNCERR_LOCAL_BUFF_TOO_SMALL. (In fact, if you pass in a NULL value for *pChangeContext* or a non-NUL value for *pChangeContextLength* that is smaller than the actual size of the change context, then this function sets **pChangeContextLength* to the actual size.)

To retrieve the change context data, set **pChangeContextLength* to at least the size passed back by the previous call. Allocate an array of the same number of HSByte values pointed to by *pChangeContext*. Call this function again and specify the same name, creator ID, and type of the closed schema database. Then the Sync Manager fills in the array with the change context data and sets **pChangeContextLength* to the number of bytes it fills in.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncDbGetSyncMode \(\)](#)

SyncDbGetColumnDefinitions Function

Purpose	Retrieves the definitions of the specified table columns.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbGetColumnDefinitions (HSByte handle, const char *tableName, UInt32 numColumns, DbColumnDefn *pColumnDefnList)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>tableName</i> The name of the table from which to retrieve column definitions.</p> <p>→ <i>numColumns</i> The number of columns in the <i>pColumnDefnList</i> array.</p> <p>↔ <i>pColumnDefnList</i> An array of <i>numColumns</i> DbColumnDefn structures that specifies the columns for which to retrieve definitions. Upon return, it receives a success or error code in each column definition's <i>errCode</i> field.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_INVALID_TABLENAME The <i>tableName</i> parameter specifies an invalid table name or a valid table name that does not exist.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p>SYNCERR_ONE_OR_MORE_FAILED Retrieval of one or more column definitions failed. Check the <i>errCode</i> field of each DbColumnDefn in the <i>pColumnDefnList</i> array for details.</p>

Schema Sync Manager API

SyncDbGetColumnDefinitions

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_TABLE_NAME_NOT_SPECIFIED

The *tableName* parameter specifies a null pointer or points to an empty character array.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

If this function returns SYNCERR_ONE_OR_MORE_FAILED, check the *errCode* field of each [*DbColumnDefn*](#) in the *pColumnDefnList* array to determine which column definitions were successfully retrieved and which were not. For successfully retrieved column definitions, the *errCode* is SYNCERR_NONE; otherwise it is set to this error code:

SYNCERR_INVALID_COLUMNID

The column ID specified in *columnID* is invalid.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[*DbColumnDefn*](#), [*SyncDbAddColumns\(\)*](#)

SyncDbGetColumnIDs Function

Purpose	Retrieves a list of all column IDs in a table.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbGetColumnIDs (HSByte handle, const char *tableName, UInt32 *pNumColumns, ColumnID *pColumnIDList)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>tableName</i> The name of the table from which to retrieve column IDs.</p> <p>↔ <i>pNumColumns</i> Specifies the number of elements that the caller allocates in the <i>pColumnIDList</i> array. The caller can pass in an address to a zero value, but it must not pass in NULL. Upon return, receives the number of column IDs that this function retrieves, if the number of elements in the <i>pColumnIDList</i> array is sufficient; or the number of column IDs in the table on the handheld, if the number of elements in the <i>pColumnIDList</i> array is insufficient.</p> <p>← <i>pColumnIDList</i> An array of <i>*pNumColumns</i> values of type ColumnID that receives the table's column IDs. The caller can specify NULL, but only if <i>*pNumColumns</i> is 0.</p>
Returns	<p>If successful, returns one of the following values:</p> <p>SYNCERR_MORE The <i>pColumnIDList</i> array is not large enough. Allocate this array at least to the size passed back in <i>pNumColumns</i> and call this function again to retrieve all column IDs in this table.</p> <p>SYNCERR_NONE The <i>pColumnIDList</i> array contains all the column IDs in this table.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p>

Schema Sync Manager API

SyncDbGetColumnIDs

SYNCERR_INVALID_FILE_HANDLE

The *handle* parameter is 0, which is an invalid value.

SYNCERR_INVALID_TABLENAME

The *tableName* parameter specifies an invalid table name or a valid table name that does not exist.

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_TABLE_NAME_NOT_SPECIFIED

The *tableName* parameter specifies a null pointer or points to an empty character array.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

This function requires that the caller allocate memory for the *pColumnIDList* array of [ColumnID](#) values, as necessary.

To retrieve the number of column IDs, set *pNumColumns* to point to a zero value and *pColumnIDList* to NULL. Specify the open database handle and table name and call this function. Then the Sync Manager retrieves the list of column IDs from the handheld, caches it, sets **pNumColumns* to the actual column ID count, and returns SYNCERR_MORE. (In fact, if you pass in a NULL value for *pColumnIDList* or a non-NULL value for *pColumnIDList* but any value for **pNumColumns* that is smaller than the column count, then this function sets **pNumColumns* to the actual count.)

To retrieve the list of column IDs, set **pNumColumns* equal to the count passed back by the previous call. Allocate an array of the same number of [ColumnID](#) values pointed to by *pColumnIDList*. Call this function again and specify the same database and table name. Then the Sync Manager fills in the array with the column IDs and sets **pNumColumns* to the number of column IDs it fills in.

Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	ColumnID , SyncDbGetColumnDefinitions()

Schema Sync Manager API

SyncDbGetColumnPropertyValue

SyncDbGetColumnPropertyValue Function

Purpose Retrieves the property value of a table column.

Declared In SyncDb.h

Prototype

```
HSError SyncDbGetColumnPropertyValue
    (HSByte handle, const char *tableName,
     ColumnID columnID,
     DbSchemaColumnProperty propID,
     UInt32 *pNumBytes, HSByte *pPropertyValue)
```

Parameters	
→ <i>handle</i>	A handle to a schema database that is open for reading. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase() .
→ <i>tableName</i>	The name of the table from which to retrieve a column property value.
→ <i>columnID</i>	The ColumnID value of the column for which to retrieve a property value.
→ <i>propID</i>	The DbSchemaColumnProperty value of the column property of which to retrieve a value.
↔ <i>pNumBytes</i>	Specifies the number of bytes that the caller allocates in the <i>pPropertyValue</i> array. The caller can pass in an address to a zero value, but it must not pass in NULL. Upon return, receives the number of bytes of the property value that this function retrieves, if the size of the <i>pPropertyValue</i> array is sufficient; or the actual length of the property value, if the size of the <i>pPropertyValue</i> buffer specified by this parameter is insufficient.
← <i>pPropertyValue</i>	An array of bytes that receives the property value. The caller can specify NULL, but only if <i>*pNumBytes</i> is 0.

Returns If successful, returns one of the following values:

SYNCERR_MORE

The *pPropertyValue* array is not large enough. Allocate this array at least to the size passed back in *pNumBytes* and call this function again to retrieve all of the property value.

SYNCERR_NONE

The *pPropertyValue* array contains all the property value.

If unsuccessful, returns one of the following error code values:

SYNCERR_BAD_ARG

One or more of the parameters passed in are invalid.

SYNCERR_INVALID_COLUMNID

The column ID specified in the *columnID* parameter is invalid.

SYNCERR_INVALID_FILE_HANDLE

The *handle* parameter is 0, which is an invalid value.

SYNCERR_INVALID_PROPID

The column property ID specified in the *propID* parameter is invalid.

SYNCERR_INVALID_TABLENAME

The *tableName* parameter specifies an invalid table name or a valid table name that does not exist.

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_TABLE_NAME_NOT_SPECIFIED

The *tableName* parameter specifies a null pointer or points to an empty character array.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

For a description of all Sync Manager error codes, see "[Schema Sync Manager Error Codes](#)" on page 207 and "[Common Sync Manager Error Codes](#)" on page 486.

Schema Sync Manager API

SyncDbGetColumnPropertyValue

Comments	This function requires that the caller allocate memory for the <i>pPropertyValue</i> byte array, as necessary. To retrieve the size of the column property value, set <i>pNumBytes</i> to point to a zero value and <i>pPropertyValue</i> to NULL. Specify the database, table name, column ID, and property ID and call this function. Then the Sync Manager retrieves the property value from the handheld, caches it, sets <i>*pNumBytes</i> to the actual size of the property value, and returns SYNCERR_MORE. (In fact, if you pass in a NULL value for <i>pPropertyValue</i> or a non-NUL value for <i>pPropertyValue</i> but any value for <i>*pNumBytes</i> that is smaller than the actual size of the property value, then this function sets <i>*pNumBytes</i> to the actual size.) To retrieve the column property value, set <i>*pNumBytes</i> equal to the size passed back by the previous call. Allocate a byte array of the same size pointed to by <i>pPropertyValue</i> . Call this function again and specify the same database, table name, column ID, and property ID. Then the Sync Manager fills in the array with the property value and sets <i>*pNumBytes</i> to the number of bytes it fills in.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	ColumnID , DbSchemaColumnProperty

SyncDbGetColumnValue Function

Purpose	Retrieves the specified bytes of a single column value for a row.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbGetColumnValue (HSByte handle, RowID rowID, ColumnID columnID, UInt32 dataOffset, UInt32 *pDataSize, HSByte *pData, UInt32 *pDataRemaining)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>rowID</i> The RowID value identifying the row for which to retrieve a column value.</p> <p>→ <i>columnID</i> The ColumnID value of the column for which to retrieve a column value.</p> <p>→ <i>dataOffset</i> An offset from the first byte in the column value from which to start retrieving data.</p> <p>↔ <i>pDataSize</i> Specifies the number of bytes of the column value to retrieve starting from the <i>dataOffset</i> position. The caller can pass in an address to a zero value, but it must not pass in NULL. Receives the number of bytes actually retrieved and stored in the <i>pData</i> buffer.</p> <p>← <i>pData</i> A pointer to an array of HSByte values that receives <i>*pDataSize</i> bytes of the column value. The caller can pass in NULL, but only if <i>*pDataSize</i> is set to 0.</p> <p>← <i>pDataRemaining</i> The number of bytes after the last one read (<i>dataOffset + *pDataSize</i>) to the end of the column value.</p>

Schema Sync Manager API

SyncDbGetColumnValue

Returns If successful, returns one of the following values:

SYNCERR_MORE

The *pData* array contains only some of the column value—that is, **pDataRemaining* > 0.

SYNCERR_NONE

The *pData* array contains all of the column value—that is, **pDataRemaining* = 0.

If unsuccessful, returns one of the following error code values:

SYNCERR_BAD_ARG

One or more of the parameters passed in are invalid.

SYNCERR_INVALID_COLUMNID

The column ID specified in the *columnID* parameter is invalid.

SYNCERR_INVALID_FILE_HANDLE

The *handle* parameter is 0, which is an invalid value.

SYNCERR_NOT_FOUND

The specified row is not present in the database.

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

To read a portion of the column value, the caller specifies an offset (*dataOffset*) from the first byte and the number of bytes to read (*pDataSize*). This function fills in the preallocated byte array with the specified number of bytes and sets **pDataSize* to the number of bytes it filled in. It also passes back the number of bytes remaining from *dataOffset* + **pDataSize* to the end of the block.

The total column value size is *dataOffset* + **pDataSize* + **pDataRemaining*.

To retrieve all of a column value of unknown size, the caller must call this function two consecutive times: once to get the size of the column value, so the caller can allocate memory for it, and again to fill in the byte array with data.

To retrieve the size of the column value, specify the following parameter values:

- *dataOffset* = 0
- **pDataSize* = 0
- *pData* = NULL

Specify the database handle, row ID, and column ID and then call this function. The Sync Manager passes back the actual size of the column value in **pDataRemaining*.

To retrieve all of the column value, set *pDataSize* to at least the size passed back by the previous call. Allocate a byte array of the same number of HSByte values pointed to by *pData*. Call this function again and specify the same database handle. Then the Sync Manager fills in the byte array with the column value and sets *pDataSize* to the number of bytes it fills in.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[RowID](#), [ColumnID](#), [SyncDbGetColumnValues\(\)](#)

Schema Sync Manager API

SyncDbGetColumnValues

SyncDbGetColumnValues Function

Purpose	Retrieves one or more column value for a row.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbGetColumnValues (HSByte handle, RowID rowID, UInt32 numColumnValues, DbColumnValue *pColumnValues)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>rowID</i> The RowID value identifying the row for which to retrieve column values.</p> <p>→ <i>numColumnValues</i> The number of column values that the caller allocates in the <i>pColumnValues</i> array.</p> <p>↔ <i>pColumnValues</i> An array of <i>numColumnValues</i> structures of type DbColumnValue that specifies the columns from which to retrieve values. Upon return, it receives a success or error code in each column value's <i>errCode</i> field and the column value and size in other fields.</p>
Returns	<p>If successful, returns one of the following values:</p> <p>SYNCERR_MORE The <i>pColumnValues</i> array is not large enough. Allocate this array at least to the size passed back in <i>numColumnValues</i> and call this function again to retrieve all of the column value. This error is also returned if any of the <i>columnData</i> values have not been retrieved in any DbColumnValue structure in the <i>pColumnValues</i> array. In this case, allocate enough memory and assign it to the <i>columnData</i> field.</p> <p>SYNCERR_NONE The <i>pColumnValues</i> array contains all the column value.</p>

If unsuccessful, returns one of the following error code values:

SYNCERR_BAD_ARG

One or more of the parameters passed in are invalid.

SYNCERR_INVALID_FILE_HANDLE

The *handle* parameter is 0, which is an invalid value.

SYNCERR_NOT_FOUND

The specified row is not present in the database.

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_ONE_OR_MORE_FAILED

Retrieval of one or more column values failed. Check the *errCode* field of each [DbColumnValue](#) in the *pColumnValues* array for details.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

For a description of all Sync Manager error codes, see "[Schema Sync Manager Error Codes](#)" on page 207 and "[Common Sync Manager Error Codes](#)" on page 486.

Comments

If this function returns SYNCERR_ONE_OR_MORE_FAILED, check the *errCode* field of each [DbColumnValue](#) in the *pColumnValues* array to determine which column values were successfully retrieved and which were not. For successfully retrieved column values, the *errCode* is SYNCERR_NONE; otherwise it is set to this error code:

SYNCERR_INVALID_COLUMNID

The column ID specified in *columnID* is invalid.

SYNCERR_MORE

The **columnData* buffer is too small to hold all the column value. Allocate at least as much memory as indicated by the value that the *dataSize* field receives and call this function again.

Schema Sync Manager API

SyncDbGetColumnValues

If in each [DbColumnValue](#) structure in the array you set the columnData field to NULL, you must also set the dataSize field to 0. However, if you set dataSize to a nonzero value, then columnData must not be NULL. In either case, if dataSize is less than the actual size of the data available for that column value, then this function passes back in the actual dataSize value and sets the errCode field to SYNCERR_MORE. From this information, you can allocate sufficient memory for the *pColumnValues* array and each columnData field, fill in the dataSize fields appropriately, initialize all other fields, and call this function again, which fills in the array with column values.

Compatibility Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [DbColumnValue](#), [SyncDbGetColumnValue\(\)](#)

SyncDbGetModifiedCategoryDefinitionList Function

Purpose	Retrieves a list of the definitions of all the modified categories in a schema database.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbGetModifiedCategoryDefinitionList (HSByte handle, UInt32 *pNumElements, DbCategoryDefn *pCategoryDefnList)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>↔ <i>pNumElements</i> Specifies the number of elements that the caller allocated in the <i>pCategoryDefnList</i> array. The caller can pass in an address to a zero value, but it must not pass in NULL. Upon return, receives the number of modified category definitions that this function retrieves, if the number of elements in the <i>pCategoryDefnList</i> array is sufficient; or the number of modified category definitions in the database on the handheld, if the number of elements in the <i>pCategoryDefnList</i> array is insufficient.</p> <p>← <i>pCategoryDefnList</i> An array of <i>*pNumElements</i> structures of type DbCategoryDefn that receives the category ID and name of each category. The caller can specify NULL, but only if <i>*pNumElements</i> is 0.</p>

Schema Sync Manager API

SyncDbGetModifiedCategoryDefinitionList

Returns If successful, returns one of the following values:

SYNCERR_MORE

The *pCategoryDefnList* array is not large enough.
Allocate this array to the size passed back in *pNumElements*
and call this function again to retrieve all modified category
definitions.

SYNCERR_NONE

The *pCategoryDefnList* array contains all the modified
category definitions.

If unsuccessful, returns one of the following error code values:

SYNCERR_BAD_ARG

One or more of the parameters passed in are invalid.

SYNCERR_INVALID_FILE_HANDLE

The *handle* parameter is 0, which is an invalid value.

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This
function operates only on schema databases.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the
handheld.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this
function does not support. See "Compatibility" below.

For a description of all Sync Manager error codes, see "[Schema Sync Manager Error Codes](#)" on page 207 and "[Common Sync Manager Error Codes](#)" on page 486.

Comments

This function requires that the caller allocate memory for the
pCategoryDefnList array of [*DbCategoryDefn*](#) structures, as
necessary.

Because you likely do not know the number of categories in the
database that have been modified since the last HotSync operation,
you can call this function to retrieve the modified category count in
**pNumElements*: pass in NULL in *pCategoryDefnList* and 0 in
**pNumElements*. This function passes back in **pNumElements*
the number of modified categories actually in the database and
returns SYNCERR_MORE.

Now that you know the number of modified categories in the database, call this function again with a *pCategoryDefnList* array and **pNumElements* value equal to the size passed back from the first call.

The first time you call this function the Sync Manager retrieves the modified category definitions from the handheld and caches them on the desktop. The cache is preserved until you call the next Sync Manager function. If the call is not for the cached data, the cache is emptied. Therefore multiple calls to this function do not incur the performance penalty of retrieving data from the handheld each time over a possibly slow connection.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[DbCategoryDefn](#), [SyncDbGetCategoryDefinitionList\(\)](#)

Schema Sync Manager API

SyncDbGetModifiedTableInfoList

SyncDbGetModifiedTableInfoList Function

Purpose	Retrieves a list of names and column counts of all the modified tables in a schema database.
Declared In	SyncDb.h
Prototype	HSError SyncDbGetModifiedTableInfoList (HSByte handle, UInt32 *pNumElements, DbTableDefn *pTableList)
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>↔ <i>pNumElements</i> Specifies the number of elements that the caller allocates in the <i>pTableList</i> array. The caller can pass in an address to a zero value, but it must not pass in NULL. Upon return, receives the number of modified table definitions that this function retrieves, if the number of elements in the <i>pTableList</i> array is sufficient; or the number of modified table definitions in the database on the handheld, if the number of elements in the <i>pTableList</i> array is insufficient.</p> <p>← <i>pTableList</i> An array of <i>*pNumElements</i> structures of type DbTableDefn that receives only the names and column counts of each modified table. This function does not fill in the <i>columnList</i> fields of these structures. The caller can specify NULL, but only if <i>*pNumElements</i> is 0.</p>
Returns	<p>If successful, returns one of the following values:</p> <p>SYNCERR_MORE The <i>pTableList</i> array is not large enough. Allocate this array to the size passed back in <i>pNumElements</i> and call this function again to retrieve all of the modified table definitions.</p> <p>SYNCERR_NONE The <i>pTableList</i> array contains all of the modified table definitions.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p>

SYNCERR_INVALID_FILE_HANDLE

The *handle* parameter is 0, which is an invalid value.

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

For a description of all Sync Manager error codes, see "[Schema Sync Manager Error Codes](#)" on page 207 and "[Common Sync Manager Error Codes](#)" on page 486.

Comments

This function requires that the caller allocate memory for the *pTableList* array of [*DbTableDefn*](#) structures, as necessary.

Because you likely do not know the number of modified tables in the database, you can call this function to retrieve the modified table count in **pNumElements*: pass in NULL in *pTableList* and 0 in **pNumElements*. This function passes back in *pTableList* the number of modified tables actually in the database and returns SYNCERR_MORE..

Now that you know the number of modified tables in the database, call this function again with a *pTableList* array and **pNumElements* value equal to the size passed back from the first call. This function fills in only the *numColumns* and *name* fields of each [*DbTableDefn*](#) structure in the array.

The first time you call this function the Sync Manager retrieves the modified table information from the handheld and caches it on the desktop. The cache is preserved until you call the next Sync Manager function. If the call is not for the cached data, the cache is emptied. Therefore multiple calls to this function do not incur the performance penalty of retrieving data from the handheld each time over a possibly slow connection.

Schema Sync Manager API

SyncDbGetModifiedTableInfoList

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [DbTableDefn](#), [SyncDbGetTableInfoList\(\)](#)

SyncDbGetRowCategory Function

Purpose	Retrieve the category membership for a row.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbGetRowCategory (HSByte handle, RowID rowID, UInt32 *pNumCategories, CategoryID *pCategoryIDList)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>rowID</i> The RowID value identifying the row for which to retrieve category membership.</p> <p>↔ <i>pNumCategories</i> Specifies the number of elements that the caller allocates in the <i>pCategoryIDList</i> array. The caller can pass in an address to a zero value, but it must not pass in NULL. Upon return, receives the number of category memberships that this function retrieves, if the number of elements in the <i>pCategoryIDList</i> array is sufficient; or the number of category memberships of the row on the handheld, if the number of elements in the <i>pCategoryIDList</i> array is insufficient.</p> <p>← <i>pCategoryIDList</i> An array of <i>*pNumCategories</i> values of type CategoryID that receives the category ID of each category to which the row belongs. The caller can specify NULL, but only if <i>*pNumCategories</i> is 0.</p>

Schema Sync Manager API

SyncDbGetRowCategory

Returns If successful, returns one of the following values:

SYNCERR_MORE

The *pCategoryIDList* array is not large enough. Allocate this array at least to the size passed back in *pNumCategories* and call this function again to retrieve all of the row's category membership list.

SYNCERR_NONE

The *pCategoryIDList* array contains all the row's category membership list.

If unsuccessful, returns one of the following error code values:

SYNCERR_BAD_ARG

One or more of the parameters passed in are invalid.

SYNCERR_INVALID_FILE_HANDLE

The *handle* parameter is 0, which is an invalid value.

SYNCERR_NOT_FOUND

The specified row is not present in the database.

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

For a description of all Sync Manager error codes, see "[Schema Sync Manager Error Codes](#)" on page 207 and "[Common Sync Manager Error Codes](#)" on page 486.

Comments

This function requires that the caller allocate memory for the *pCategoryIDList* array of *CategoryID* values, as necessary.

To retrieve the number of categories that a row belongs to, set *pNumCategories* to point to a zero value and *pCategoryIDList* to NULL. Specify the database handle and row ID and call this function. Then the Sync Manager retrieves the list of category IDs from the handheld, caches it, sets **pNumCategories* to the actual category count, and returns SYNCERR_MORE. (In fact, if you pass in

a NULL value for *pCategoryIDList* or a non-NULL value for *pCategoryIDList* but any value for **pNumCategories* that is smaller than the category count, then this function sets **pNumCategories* to the actual count.)

To retrieve the list of category IDs, set **pNumCategories* to at least the count passed back by the previous call. Allocate an array of the same number of [CategoryID](#) values pointed to by *pCategoryIDList*. Call this function again and specify the same database handle and row ID. Then the Sync Manager fills in the array with the category IDs and sets **pNumCategories* to the number of category IDs it fills in.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[CategoryID](#), [SyncDbGetCategoryDefinitionList](#)

Schema Sync Manager API

SyncDbGetRowCountInTable

SyncDbGetRowCountInTable Function

Purpose	Retrieves the total number of rows in a table.
Declared In	SyncDb.h
Prototype	HSError SyncDbGetRowCountInTable (HSByte <i>handle</i> , HSBool <i>excludeDelete</i> , const char * <i>tableName</i> , UInt32 * <i>pRowCount</i>)
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>excludeDelete</i> If TRUE, do not include in the count rows that are marked deleted. If FALSE, include all rows in the count.</p> <p>→ <i>tableName</i> The name of the table from which to retrieve the row count. If you specify NULL, this function retrieves the total number of rows in all the tables in the database.</p> <p>← <i>pRowCount</i> A pointer to the number of rows in the table.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_INVALID_TABLENAME The <i>tableName</i> parameter specifies an invalid table name or a valid table name that does not exist.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p>SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.</p>

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

Schema Sync Manager API

SyncDbGetSyncMode

SyncDbGetSyncMode Function

Purpose	Determines the synchronization mode for a schema database.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbGetSyncMode (HSByte handle, DbSyncMode *pSyncMode, UInt32 nChangeContextLength, HSByte *pChangeContext)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>← <i>pSyncMode</i> A pointer to a DbSyncMode structure, which receives the type of synchronization that the Sync Manager determines should occur for each sync atoms.</p> <p>→ <i>nChangeContextLength</i> Specifies the length of the change context that the caller passes in the buffer pointed to by the <i>pChangeContext</i> parameter. You must specify 0 if <i>pChangeContext</i> is NULL.</p> <p>→ <i>pChangeContext</i> A buffer that specifies the change context. Most conduits do not need to specify a change context. To use the change context that Sync Manager cached on the current desktop, specify NULL and set <i>nChangeContextLength</i> to 0.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_LOCAL_MEM This function was unable to allocate desktop memory to read the change context from the Sync Manager's cache.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p>

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

For a description of all Sync Manager error codes, see "[Schema Sync Manager Error Codes](#)" on page 207 and "[Common Sync Manager Error Codes](#)" on page 486.

Comments

A conduit must call this function for each schema database it synchronizes to determine *for each sync atom* whether it can perform a fast sync, slow sync, or a fast sync after a purge.

IMPORTANT: The `eFast` or `eSlow` value in `CSyncProperties::m_SyncType`, which HotSync Manager passes back through your conduit's `OpenConduit()` entry point, is based only on whether the last HotSync operation was with the current desktop. Do not rely on this value alone to determine whether to perform a fast or slow sync with a schema database. Instead, call `SyncDbGetSyncMode()` to take full advantage of the extra change tracking information that is available only in schema databases and not in classic and extended databases.

This function determines the sync mode for each sync atom by comparing the change context that the Sync Manager cached on the desktop during the last HotSync operation with the one it obtains from the handheld. Alternatively, you can pass in the change context via the `pChangeContext` parameter that you received from a call to `SyncDbGetChangeContext()` and cached (possibly on a networked server) during a previous HotSync operation.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[DbSyncMode](#), [SyncDbGetChangeContext\(\)](#)

Schema Sync Manager API

SyncDbGetTableCount

SyncDbGetTableCount Function

Purpose	Retrieves the total number of tables in a schema database.
Declared In	SyncDb.h
Prototype	<code>HSError SyncDbGetTableCount (HSByte handle, UInt32 *pTableCount)</code>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>← <i>pTableCount</i> A pointer to the number of tables in the database.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p>SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.</p> <p>SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.</p> <p>For a description of all Sync Manager error codes, see “Schema Sync Manager Error Codes” on page 207 and “Common Sync Manager Error Codes” on page 486.</p>
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncDbGetTableInfoList()

SyncDbGetTableInfo Function

Purpose	Retrieves the number of columns in a table.
Declared In	SyncDb.h
Prototype	HSError SyncDbGetTableInfo (HSByte <i>handle</i> , DbTableDefn * <i>pTableDefn</i>)
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>↔ <i>pTableDefn</i> A pointer to a DbTableDefn structure that specifies the name of the table for which to retrieve the column count. Upon return, it receives the table defintion from the handheld with only the numColumns field filled in.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_INVALID_TABLENAME The <i>pTableDefn.name</i> field specifies an invalid table name or a valid table name that does not exist.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p>SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.</p> <p>SYNCERR_TABLE_NAME_NOT_SPECIFIED The <i>pTableDefn.name</i> field points to an empty character array.</p> <p>SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.</p>

Schema Sync Manager API

SyncDbGetTableInfo

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Compatibility Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [DbTableDefn](#), [SyncDbGetTableSchema \(\)](#)

SyncDbGetTableInfoList Function

Purpose Retrieves the number of columns in each of the specified tables.

Declared In SyncDb.h

Prototype HSError SyncDbGetTableInfoList (HSByte handle,
 UInt32 *pNumElements, DbTableDefn *pTableList)

Parameters → *handle*

A handle to a schema database that is open for reading. This handle is returned by [SyncDbOpenDatabase\(\)](#) or [SyncDbCreateDatabase\(\)](#).

↔ *pNumElements*

Specifies the number of elements that the caller allocates in the *pTableList* array. The caller can pass in an address to a zero value, but it must not pass in NULL. Upon return, receives the number of table definitions that this function retrieves, if the number of elements in the *pTableList* array is sufficient; or the number of table definitions in the database on the handheld, if the number of elements in the *pTableList* array is insufficient.

↔ *pTableList*

An array of **pNumElements* structures of type [DbTableDefn](#) that specifies the names of the tables for which to get the column counts. Upon return, receives only the column counts of the specified tables. This function does not fill in the *columnList* fields of these structures. The caller can specify NULL, but only if **pNumElements* is 0.

Returns If successful, returns one of the following values:

SYNCERR_NONE

The *pTableList* array contains all the table definitions.

SYNCERR_MORE

The *pTableList* array is not large enough. Allocate this array to the size passed back in *pNumElements* and call this function again to retrieve all table definitions.

If unsuccessful, returns one of the following error code values:

SYNCERR_BAD_ARG

One or more of the parameters passed in are invalid.

SYNCERR_INVALID_FILE_HANDLE

The *handle* parameter is 0, which is an invalid value.

Schema Sync Manager API

SyncDbGetTableInfoList

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

For a description of all Sync Manager error codes, see "[Schema Sync Manager Error Codes](#)" on page 207 and "[Common Sync Manager Error Codes](#)" on page 486.

Comments

This function requires that the caller allocate memory for the *pTableList* array of [*DbTableDefn*](#) structures, as necessary. If you want to retrieve the column counts of all tables in the database and you do not know how many there are, you can call this function to retrieve the table count in **pNumElements*: pass in NULL in *pTableList* and 0 in **pNumElements*. This function passes back in *pTableList* the number of tables actually in the database and returns SYNCERR_MORE. Alternatively, you can call [*SyncDbGetTableCount\(\)*](#) to retrieve only the number of tables.

Now that you know the number of tables in the database (or knew it already), call this function again with a *pTableList* array and **pNumElements* value equal to the size passed back from the first call. This function fills in only the *numColumns* and *name* fields of each [*DbTableDefn*](#) structure in the array.

The first time you call this function the Sync Manager retrieves the table information from the handheld and caches it on the desktop. The cache is preserved until you call the next Sync Manager function. If the call is not for the cached data, the cache is emptied. Therefore multiple calls to this function do not incur the performance penalty of retrieving data from the handheld each time over a possibly slow connection.

Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	DbTableDefn , SyncDbGetTableCount() , SyncDbGetModifiedTableInfoList() , SyncDbGetTableSchema()

Schema Sync Manager API

SyncDbGetTableSchema

SyncDbGetTableSchema Function

Purpose	Retrieves the schema for a table, including the definitions and built-in properties for all of the table's columns.
Declared In	SyncDb.h
Prototype	HSError SyncDbGetTableSchema (HSByte <i>handle</i> , DbTableDefn * <i>pTableDefn</i>)
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>↔ <i>pTableDefn</i> A pointer to a DbTableDefn structure that specifies the name of the table for which to retrieve the schema. Upon return, it receives the table defintion from the handheld.</p>
Returns	If successful, returns one of the following values: SYNCERR_MORE The <i>pTableDefn.columnList</i> array is not large enough. Allocate this array equal to the size passed back in <i>pTableDefn.numColumns</i> and call this function again to retrieve all column definitions. SYNCERR_NONE The <i>pTableDefn.columnList</i> array contains all the column defintions. If unsuccessful, returns one of the following error code values: SYNCERR_BAD_ARG One or more of the parameters passed in are invalid. SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value. SYNCERR_INVALID_TABLENAME The <i>pTableDefn.name</i> field specifies an invalid table name or a valid table name that does not exist. SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_TABLE_NAME_NOT_SPECIFIED

The *pTableDefn.name* field points to an empty character array.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

For a description of all Sync Manager error codes, see "[Schema Sync Manager Error Codes](#)" on page 207 and "[Common Sync Manager Error Codes](#)" on page 486.

Comments

This function requires that the caller allocate memory for the *pTableDefn.columnList* array of [*DbColumnDefn*](#) structures, as necessary.

If you do not know the number of columns in the table, you can call this function to retrieve the column count in

**pTableDefn.numColumns*: pass in NULL in *pTableDefn.columnList* and 0 in **pTableDefn.numColumns*. This function passes back in *pTableDefn.numColumns* the number of columns actually in this table and returns SYNCERR_MORE..

Now that you know the number of columns in the table (or knew it already), call this function again with a *pTableDefn.columnList* array and **pTableDefn.numColumns* value at least the size passed back from the first call. Upon return, this function passes back in **pTableDefn.numColumns* the number of columns actually in this table and the column definitions in the *pTableDefn.columnList* array.

The first time you call this function the Sync Manager retrieves the column definitions from the handheld and caches them on the desktop. The cache is preserved until you call the next Sync Manager function. If the call is not for the cached data, the cache is emptied. Therefore multiple calls to this function do not incur the performance penalty of retrieving data from the handheld each time over a possibly slow connection.

Schema Sync Manager API

SyncDbGetTableSchema

Compatibility Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [DbTableDefn](#), [DbColumnDefn](#),
[SyncDbGetColumnDefinitions\(\)](#), [SyncDbGetTableInfo\(\)](#)

SyncDbIsRowInCategory Function

Purpose Determines whether a row is a member of the specified categories, depending on the given match mode criteria.

Declared In SyncDb.h

Prototype

```
HSError SyncDbIsRowInCategory (HSByte handle,
                                RowID rowID, UInt32 numCategories,
                                const CategoryID *pCategoryIDList,
                                DbMatchModeType matchMode,
                                HSBool *pInCategory)
```

Parameters

→ *handle*
A handle to a schema database that is open for reading. This handle is returned by [SyncDbOpenDatabase\(\)](#) or [SyncDbCreateDatabase\(\)](#).

→ *rowID*
The [RowID](#) value identifying the row for which to determine whether it is a member of the categories in the *pCategoryIDList* array.

→ *numCategories*
The number of category IDs in the *pCategoryIDList* array.

→ *pCategoryIDList*
An array of *numCategories* values of type [CategoryID](#) that specifies the categories that the row must be in for this function to set **pInCategory* to true.

→ *matchMode*
One of the following [DbMatchModeType](#) values that specifies how the categories in the *pCategoryIDList* array must match a given row's category membership for this function to set the *pInCategory* upon return:

DbMatchAny
(OR) Set *pInCategory* to true if the row is in *any* of the categories specified in the *pCategoryIDList* array.

DbMatchAll
(AND) Set *pInCategory* to true if the row is in *all* of the categories specified in the *pCategoryIDList* array *and is possibly in others*.

Schema Sync Manager API

SyncDbIsRowInCategory

DbMatchExact

Set *pInCategory* to true if the row is in *all* of the categories specified in the *pCategoryIDList* array and is in no others.

← *pInCategory*

If true, the row matches the categories in the *pCategoryIDList* parameter according to the specified match mode. If false, the row does not match.

Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following error code values: SYNCERR_BAD_ARG One or more of the parameters passed in are invalid. SYNCERR_INVALID_CATEGORYID The allowed number of categories has been exceeded, or a category ID doesn't correspond to a defined category. SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value. SYNCERR_INVALID_MATCH_MODE The <i>matchMode</i> parameter specifies an invalid match mode. Specify any one of the values described in " Category Match Modes " on page 30. SYNCERR_NOT_FOUND The specified row is not present in the database. SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases. SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld. SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below. For a description of all Sync Manager error codes, see " Schema Sync Manager Error Codes " on page 207 and " Common Sync Manager Error Codes " on page 486.
----------------	--

Comments	To check whether a row has no category membership (that is, it belongs to the “Unfiled” category), set <i>numCategories</i> to 0 and <i>pCategoryIDList</i> to NULL.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	CategoryID , DbMatchModeType , SyncDbGetRowCategory()

Schema Sync Manager API

SyncDbModifyCategory

SyncDbModifyCategory Function

Purpose	Modify a category in a schema database.
Declared In	SyncDb.h
Prototype	HSError SyncDbModifyCategory (HSByte <i>handle</i> , const DbCategoryDefn * <i>pCategoryDefn</i>)
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>pCategoryDefn</i> A pointer to a DbCategoryDefn structure that specifies the category ID of the category to modify and the new category name.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_CATEGORYID The allowed number of categories has been exceeded, or a category ID doesn't correspond to a defined category.</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_NAME_ALREADY_EXISTS The category name specified in the name field already exists.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p>SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.</p> <p>SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.</p>

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [DbCategoryDefn](#)

Schema Sync Manager API

SyncDbMoveRowsInCategory

SyncDbMoveRowsInCategory Function

Purpose	Determines all the rows that are members of the specified categories, depending on the given match mode criteria, and moves them to a single target category.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbMoveRowsInCategory (HSByte handle, UInt32 numCategories, const CategoryID *pCategoryIDList, DbMatchModeType matchMode, CategoryID toCategory)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>numCategories</i> The number of category IDs in the <i>pCategoryIDList</i> array.</p> <p>→ <i>pCategoryIDList</i> An array of <i>numCategories</i> values of type CategoryID that specifies the categories that rows must be in for this function to move them to the <i>toCategory</i> category.</p> <p>→ <i>matchMode</i> One of the following DbMatchModeType values that specifies how the categories in the <i>pCategoryIDList</i> array must match rows' category memberships for this function to move them to the <i>toCategory</i> category upon return:</p> <p> DbMatchAny (OR) Replace category membership for rows that are in <i>any</i> of the categories specified in the <i>pCategoryIDList</i> array.</p> <p> DbMatchAll (AND) Replace category membership for rows that are in <i>all</i> of the categories specified in the <i>pCategoryIDList</i> array <i>and are possibly in others</i>.</p> <p> DbMatchExact Replace category membership for rows that are in <i>all</i> of the categories specified in the <i>pCategoryIDList</i> array <i>and are in no others</i>.</p>

→ *toCategory*

The [CategoryID](#) value of a single target category to which to move the rows, if any match.

Returns

If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following error code values:

SYNCERR_BAD_ARG

One or more of the parameters passed in are invalid.

SYNCERR_INVALID_CATEGORYID

A category ID doesn't correspond to a defined category.

SYNCERR_INVALID_FILE_HANDLE

The *handle* parameter is 0, which is an invalid value.

SYNCERR_INVALID_MATCH_MODE

The *matchMode* parameter specifies an invalid match mode. Specify only values described in “[Category Match Modes](#)” on page 30.

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

You can also move row membership from no membership (“Unfiled”) to membership in a single category by specifying both of the following:

- a valid category ID value for the *toCategory* parameter
- NULL for the *pCategoryIDList* parameter and 0 for *numCategories*. In this case, the *matchMode* parameter is ignored.

Schema Sync Manager API

SyncDbMoveRowsInCategory

This function might perform no action if either of the following is true:

- None of the category IDs in *pCategoryIDList* are valid and the match mode criteria value is DbMatchAny.
- Any of the category IDs in *pCategoryIDList* are not valid and the match mode criteria value is either DbMatchAll or DbMatchExact.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [CategoryID](#), “[Category Match Modes](#)” on page 30,
[SyncDbRemoveCategoryFromAllRows\(\)](#)

SyncDbOpenDatabase Function

Purpose	Opens an existing schema database on the handheld.
Declared In	<code>SyncDb.h</code>
Prototype	<code>HSError SyncDbOpenDatabase (const char *pName, UInt32 creator, UInt16 openMode, DbShareModeType shareMode, HSByte *pHandle)</code>
Parameters	<p>→ <i>pName</i> The database name.</p> <p>→ <i>creator</i> The database creator ID.</p> <p>→ <i>openMode</i> A nonexclusive combination of the values defined in “Database Open Modes” on page 37. This parameter specifies the access mode in which to open the database—read-only, read/write, and show private records.</p> <p>→ <i>shareMode</i> One of the values defined in “Database Share Modes” on page 38. This parameter must specify only one share mode in which to open the database—share for read-only access, write access, or do not share.</p> <p>← <i>pHandle</i> A handle to the open schema database.</p>
Returns	<p>If successful, returns <code>SYNCERR_NONE</code>.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p><code>SYNCERR_ACCESS_DENIED</code> Access is denied by the Authorization Manager on the handheld or the database cannot be unencrypted.</p> <p><code>SYNCERR_BAD_ARG</code> One or more of the parameters passed in are invalid.</p> <p><code>SYNCERR_FILE_ALREADY_OPEN</code> The specified database must be closed before you call this function.</p> <p><code>SYNCERR_NOT_FOUND</code> The specified schema database is not on the handheld.</p>

Schema Sync Manager API

SyncDbOpenDatabase

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

The Sync Manager allows multiple schema databases to be open at once, unlike classic databases. Therefore you need to close schema databases only before deleting them or before exiting your conduit. However, because of the overhead incurred by an open database, you should close a schema database as soon as you no longer need it open.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

“[Database Open Modes](#)” on page 37, “[Database Share Modes](#)” on page 38, [SyncDbCloseDatabase\(\)](#)

SyncDbPurgeAllRowsInTable Function

Purpose Removes all the rows that are marked as deleted or archived in a table.

Declared In SyncDb.h

Prototype HSError SyncDbPurgeAllRowsInTable (HSByte *handle*,
const char **tableName*)

Parameters
→ *handle*
A handle to a schema database that is open for writing. This handle is returned by [SyncDbOpenDatabase\(\)](#) or [SyncDbCreateDatabase\(\)](#).

→ *tableName*
The name of the table from which to remove rows. If you specify NULL, this function removes all deleted or archived rows in *all* tables in the database.

Returns If successful, returns SYNCERR_NONE.
If unsuccessful, returns one of the following error code values:
SYNCERR_BAD_ARG
One or more of the parameters passed in are invalid.
SYNCERR_INVALID_FILE_HANDLE
The *handle* parameter is 0, which is an invalid value.
SYNCERR_INVALID_TABLENAME
The *tableName* parameter specifies an invalid table name or a valid table name that does not exist.
SYNCERR_NOT_FOUND
The specified table is not present in the database.
SYNCERR_NOT_SCHEMADB
The *handle* parameter refers to a nonschema database. This function operates only on schema databases.
SYNCERR_REMOTE_BAD_ARG
One or more invalid parameters has been passed to the handheld.
SYNCERR_TABLE_NAME_NOT_SPECIFIED
The *tableName* parameter specifies a null pointer or points to an empty character array.

Schema Sync Manager API

SyncDbPurgeAllRowsInTable

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments	For all rows in a table that have their delete bit set (DbRowData .changeFlags has dbRowDeleted or dbRowArchived set), this function immediately disposes of the rows’ data, and it removes the rows’ entries in the database header.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncDbDeleteRow() , SyncDbDeleteRows() , SyncDbDeleteRowsInCategory() , SyncDbRemoveRow() , SyncDbRemoveRows() , SyncDbRemoveSecretRowsInTable() , SyncDbRemoveTable()

SyncDbReadModifiedRowInfoList Function

Purpose Reads information about all the rows that match the specified criteria and have been modified since the last HotSync operation.

Declared In SyncDb.h

Prototype

```
HSError SyncDbReadModifiedRowInfoList
    (HSByte handle, const char *pTableName,
     UInt32 numCategories,
     const CategoryID *pCategoryIDList,
     DbMatchModeType matchMode, UInt32 *pNumRows,
     DbRowData **ppRowInfoList)
```

Parameters

→ *handle*

A handle to a schema database that is open for reading. This handle is returned by [SyncDbOpenDatabase\(\)](#) or [SyncDbCreateDatabase\(\)](#).

→ *pTableName*

A pointer to a buffer containing the name of a table. If you specify NULL, this function retrieves row information for modified rows in all tables in the database.

→ *numCategories*

The number of category IDs in the *pCategoryIDList* array.

→ *pCategoryIDList*

An array of *numCategories* values of type [CategoryID](#) that specifies the categories that modified rows must be in for this function to retrieve information about them. If you specify 0, this function does not filter modified rows based on category membership.

→ *matchMode*

One of the following [DbMatchModeType](#) values that specifies how the categories in the *pCategoryIDList* array must match modified rows' category membership for this function to read information about them:

DbMatchAny

(OR) Read information about modified rows that are in *any* of the categories specified in the *pCategoryIDList* array.

Schema Sync Manager API

SyncDbReadModifiedRowInfoList

DbMatchAll

(AND) Read information about modified rows that are in *all* of the categories specified in the *pCategoryIDList* array *and are possibly in others.*

DbMatchExact

Read information about modified rows that are in *all* of the categories specified in the *pCategoryIDList* array *and are in no others.*

← *pNumRows*

A pointer to the number of rows in the *ppRowInfoList* array. This is the number of modified rows for which information is read or is available. Do not pass in NULL.

← *ppRowInfoList*

A pointer to the address of an array of **pNumRows* structures of type [DbRowData](#) that receives information about the matching modified rows. This function allocates memory for these structures, but the caller must call [SyncDbReleaseStorage\(\)](#) to release the allocated memory.

Returns

If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following error code values:

SYNCERR_BAD_ARG

One or more of the parameters passed in are invalid.

SYNCERR_INVALID_CATEGORYID

A specified category ID doesn't correspond to a defined category.

SYNCERR_INVALID_FILE_HANDLE

The *handle* parameter is 0, which is an invalid value.

SYNCERR_INVALID_MATCH_MODE

The *matchMode* parameter specifies an invalid match mode. Specify any one of the values described in “[Category Match Modes](#)” on page 30.

SYNCERR_INVALID_TABLENAME

The *pTableName* parameter specifies an invalid table name or a valid table name that does not exist.

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

This function retrieves the data set defined by dbRowInfoDataSet in “[Row Data Sets](#)” on page 42 and allocates memory for the data for each row. When you no longer need to access the row data, you must pass the value of *ppRowInfoList to [SyncDbReleaseStorage\(\)](#) to release the allocated memory.

Each [DbRowData](#).dataSetRetrieved field is set to dbRowInfoDataSet, unless an error occurs, in which case it is set to dbRowEmptyDataSet and the errCode field is set to indicate the error.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[DbRowData](#), [SyncDbReleaseStorage\(\)](#), “[Row Data Sets](#)” on page 42, [SyncDbReadRowsByRowInfo\(\)](#), [SyncDbReadRowInfoList\(\)](#)

Schema Sync Manager API

SyncDbReadModifiedRows

SyncDbReadModifiedRows Function

Purpose Reads all of a row's data—the attributes, category membership, and all column values—for all rows that match the specified criteria and have been modified since the last HotSync operation.

Declared In SyncDb.h

Prototype HSError SyncDbReadModifiedRows (HSByte *handle*,
 const char **pTableName*, UInt32 *numCategories*,
 const CategoryID **pCategoryIDList*,
 DbMatchModeType *matchMode*, UInt32 **pNumRows*,
 DbRowData ***ppRows*)

Parameters

→ *handle*
A handle to a schema database that is open for reading. This handle is returned by a call to [SyncDbOpenDatabase\(\)](#) or [SyncDbCreateDatabase\(\)](#).

→ *pTableName*
A pointer to a buffer containing the name of a table. If you specify NULL, this function retrieves row data for modified rows in all tables in the database.

→ *numCategories*
The number of category IDs in the *pCategoryIDList* array.

→ *pCategoryIDList*
An array of *numCategories* values of type [CategoryID](#) that specifies the categories that modified rows must be in for this function to retrieve their row data. If you specify 0, this function does not filter modified rows based on category membership.

→ *matchMode*

One of the following [DbMatchModeType](#) values that specifies how the categories in the *pCategoryIDList* array must match modified rows' category membership for this function to retrieve their row data:

DbMatchAny

(OR) Read all data for modified rows that are in *any* of the categories specified in the *pCategoryIDList* array.

DbMatchAll

(AND) Read all data for modified rows that are in *all* of the categories specified in the *pCategoryIDList* array *and are possibly in others*.

DbMatchExact

Read all data for modified rows that are in *all* of the categories specified in the *pCategoryIDList* array *and are in no others*.

← *pNumRows*

A pointer to the number of rows in the *ppRows* array. This is the number of modified rows for which data is read or is available. Do not pass in NULL.

← *ppRows*

A pointer to the address of an array of **pNumRows* structures of type [DbRowData](#) that receives row data for matching modified rows. This function allocates memory for these structures, but the caller must call [SyncDbReleaseStorage\(\)](#) to release the allocated memory.

Schema Sync Manager API

SyncDbReadModifiedRows

- Returns** If successful, returns SYNCERR_NONE.
- If unsuccessful, returns one of the following error code values:
- SYNCERR_BAD_ARG
 - One or more of the parameters passed in are invalid.
 - SYNCERR_INVALID_CATEGORYID
 - The allowed number of categories has been exceeded, or a category ID doesn't correspond to a defined category.
 - SYNCERR_INVALID_HANDLE
 - The *handle* parameter is 0, which is an invalid value.
 - SYNCERR_INVALID_MATCH_MODE
 - The *matchMode* parameter specifies an invalid match mode. Specify only values described in “[Category Match Modes](#)” on page 30.
 - SYNCERR_INVALID_TABLENAME
 - The *pTableName* parameter specifies an invalid table name or a valid table name that does not exist.
 - SYNCERR_NOT_SCHEMADB
 - The *handle* parameter refers to a nonschema database. This function operates only on schema databases.
 - SYNCERR_ONE_OR_MORE_FAILED
 - Retrieval of one or more rows' data failed. Check the *errCode* field of each [DbRowData](#) in the *ppRows* array for details.
 - SYNCERR_REMOTE_BAD_ARG
 - One or more invalid parameters has been passed to the handheld.
 - SYNCERR_UNKNOWN_REQUEST
 - The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.
- For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments	<p>This function retrieves the data set defined by dbRowAllDataSet in “Row Data Sets” on page 42 and allocates memory for the data for each row. When you no longer need to access the row data, you must pass the value of *ppRowInfoList to SyncDbReleaseStorage() to release the allocated memory.</p> <p>Each DbRowData.dataSetRetrieved field is set to dbRowAllDataSet, if the row is successfully retrieved. However, if this function returns SYNCERR_ONE_OR_MORE_FAILED, check the errCode field of each DbRowData structure in the ppRows array to determine which rows’ values were successfully retrieved and which were not. For successfully retrieved row data, the errCode is SYNCERR_NONE; otherwise it is set to this error code:</p> <p>SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.</p> <p>If an error occurs for a row, that row’s RowData.errCode field indicates the error and its RowData.dataSetRetrieved field indicates which data set was successfully retrieved. For example, if an error occurred and dataSetRetrieved is (dbRowInfoDataSet dbRowCategoriesDataSet dbRowColumnInfoDataSet), then the error occurred while retrieving the data set defined by dbRowColumnValuesDataSet.</p>
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	DbRowData , SyncDbReleaseStorage() , “ Row Data Sets ” on page 42, SyncDbReadRows()

Schema Sync Manager API

SyncDbReadOpenDatabaseInfo

SyncDbReadOpenDatabaseInfo Function

Purpose	Retrieves information about an open schema database.
Declared In	<code>SyncDb.h</code>
Prototype	<code>HSError SyncDbReadOpenDatabaseInfo (HSByte handle, HSByte bOptFlags, DBDatabaseInfo *pDatabaseInfo)</code>
Parameters	<p>→ <code>handle</code> A handle to a schema database that is open for reading. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <code>bOptFlags</code> Specifies additional information to retrieve about the database. Pass in a combination of one or more of the following values:</p> <ul style="list-style-type: none"><code>SYNC_DB_INFO_OPT_GET_SCHEMADB_FIELDS</code> Retrieve encoding, table count, and display name fields in section 3 of the DBDatabaseInfo structure.<code>SYNC_DB_INFO_OPT_GET_ATTRIBUTES</code> Retrieve database attribute information.<code>SYNC_DB_INFO_OPT_GET_SIZE</code> Retrieve count and data size information. <p>These values are defined in “Database Information Retrieval Option for SyncDbReadOpenDatabaseInfo()” on page 34 and “Database Information Retrieval Options” on page 412.</p> <p>← <code>pDatabaseInfo</code> A pointer to a DBDatabaseInfo structure that receives information about the database. Do not pass in NULL.</p>

Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following error code values: SYNCERR_BAD_ARG One or more of the parameters passed in are invalid. SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value. SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases. SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld. SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	DBDatabaseInfo , SyncDbOpenDatabase() , SyncDbFindDatabase() , SyncDbFindDatabaseByTypeCreator()

Schema Sync Manager API

SyncDbReadRow

SyncDbReadRow Function

Purpose	Reads all of a row's data—the attributes, category membership, and all column values.
Declared In	SyncDb.h
Prototype	HSError SyncDbReadRow (HSByte <i>handle</i> , RowID <i>rowID</i> , DbRowData ** <i>ppRow</i>)
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading. This handle is returned by a call to SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>rowID</i> The RowID value identifying the row to read.</p> <p>← <i>ppRow</i> A pointer to the address of a DbRowData structure that receives all of the row's data. This function allocates memory for this structure, but the caller must call SyncDbReleaseStorage() to release the allocated memory.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_NOT_FOUND The specified row is not present in the database.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p>SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.</p> <p>SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.</p>

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments	This function retrieves the data set defined by dbRowAllDataSet in “ Row Data Sets ” on page 42 and allocates memory for the data for each row. When you no longer need to access the row data, you must pass the value of *ppRowInfoList to SyncDbReleaseStorage() to release the allocated memory.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	DbRowData , SyncDbReleaseStorage()

Schema Sync Manager API

SyncDbReadRowInfo

SyncDbReadRowInfo Function

Purpose	Reads information about a row.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbReadRowInfo (HSByte handle, RowID rowID, DbRowData **ppRowInfo)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>rowID</i> The RowID value identifying the row for which to retrieve information.</p> <p>← <i>ppRowInfo</i> A pointer to the address of a DbRowData structure that receives information about the row. This function allocates memory for this structure, but the caller must call SyncDbReleaseStorage() to release the allocated memory.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_NOT_FOUND The specified row is not present in the database.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p>SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.</p> <p>SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.</p>

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments	This function retrieves the data set defined by dbRowInfoDataSet in “ Row Data Sets ” on page 42 and allocates memory for the data for each row. When you no longer need to access the row data, you must pass the value of *ppRowInfoList to SyncDbReleaseStorage() to release the allocated memory.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	DbRowData , SyncDbReleaseStorage() , “ Row Data Sets ” on page 42

Schema Sync Manager API

SyncDbReadRowInfoList

SyncDbReadRowInfoList Function

Purpose Reads information about all the rows that match the specified criteria.

Declared In SyncDb.h

Prototype

```
HSError SyncDbReadRowInfoList (HSByte handle,
                               const char *pTableName, UInt32 numCategories,
                               const CategoryID *pCategoryIDList,
                               DbMatchModeType matchMode, UInt32 *pNumRows,
                               DbRowData **ppRowInfoList)
```

Parameters

→ *handle*
A handle to a schema database that is open for reading. This handle is returned by [SyncDbOpenDatabase\(\)](#) or [SyncDbCreateDatabase\(\)](#).

→ *pTableName*
A pointer to a buffer containing the name of a table. If you specify NULL, this function retrieves row information for rows in all tables in the database.

→ *numCategories*
The number of category IDs in the *pCategoryIDList* array.

→ *pCategoryIDList*
An array of *numCategories* values of type [CategoryID](#) that specifies the categories that rows must be in for this function to retrieve information about them. If you specify 0, this function does not filter rows based on category membership.

→ *matchMode*
One of the following [DbMatchModeType](#) values that specifies how the categories in the *pCategoryIDList* array must match rows' category membership for this function to read information about them:

DbMatchAny
(OR) Read information about rows that are in *any* of the categories specified in the *pCategoryIDList* array.

DbMatchAll

(AND) Read information about rows that are in *all* of the categories specified in the *pCategoryIDList* array *and are possibly in others.*

DbMatchExact

Read information about rows that are in *all* of the categories specified in the *pCategoryIDList* array *and are in no others.*

← *pNumRows*

A pointer to the number of rows in the *ppRowInfoList* array. This is the number of rows for which information is read or is available. Do not pass in NULL.

← *ppRowInfoList*

A pointer to the address of an array of **pNumRows* structures of type [DbRowData](#) that receives information about the matching rows. This function allocates memory for these structures, but the caller must call [SyncDbReleaseStorage\(\)](#) to release the allocated memory.

Returns If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following error code values:

SYNCERR_BAD_ARG

One or more of the parameters passed in are invalid.

SYNCERR_INVALID_CATEGORYID

A specified category ID doesn't correspond to a defined category.

SYNCERR_INVALID_FILE_HANDLE

The *handle* parameter is 0, which is an invalid value.

SYNCERR_INVALID_MATCH_MODE

The *matchMode* parameter specifies an invalid match mode. Specify only values described in “[Category Match Modes](#)” on page 30.

SYNCERR_INVALID_TABLENAME

The *pTableName* parameter specifies an invalid table name or a valid table name that does not exist.

Schema Sync Manager API

SyncDbReadRowInfoList

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_TABLE_NAME_NOT_SPECIFIED

The *pTableName* parameter points to an empty character array.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

For a description of all Sync Manager error codes, see "[Schema Sync Manager Error Codes](#)" on page 207 and "[Common Sync Manager Error Codes](#)" on page 486.

Comments

This function retrieves the data set defined by *dbRowInfoDataSet* in "[Row Data Sets](#)" on page 42 and allocates memory for the data for each row. When you no longer need to access the row data, you must pass the value of **ppRowInfoList* to [SyncDbReleaseStorage\(\)](#) to release the allocated memory.

Each [DbRowData](#).*dataSetRetrieved* field is set to *dbRowInfoDataSet*, unless an error occurs, in which case it is set to *dbRowEmptyDataSet* and the *errCode* field is set to indicate the error.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[DbRowData](#), [SyncDbReleaseStorage\(\)](#), "[Row Data Sets](#)" on page 42

SyncDbReadRows Function

Purpose Reads all of a row's data—the attributes, category membership, and all column values—for all rows that match the specified criteria.

Declared In SyncDb.h

Prototype

```
HSError SyncDbReadRows (HSByte handle,
    const char *pTableName, UInt32 numCategories,
    const CategoryID *pCategoryIDList,
    DbMatchModeType matchMode, UInt32 *pNumRows,
    DbRowData **ppRows)
```

Parameters

→ *handle*
A handle to a schema database that is open for reading. This handle is returned by a call to [SyncDbOpenDatabase\(\)](#) or [SyncDbCreateDatabase\(\)](#).

→ *pTableName*
A pointer to a buffer containing the name of a table. If you specify NULL, this function retrieves row data for rows in all tables in the database.

→ *numCategories*
The number of category IDs in the *pCategoryIDList* array.

→ *pCategoryIDList*
An array of *numCategories* values of type [CategoryID](#) that specifies the categories that rows must be in for this function to retrieve their row data. If you specify 0, this function does not filter rows based on category membership.

→ *matchMode*
One of the following [DbMatchModeType](#) values that specifies how the categories in the *pCategoryIDList* array must match rows' category membership for this function to retrieve their row data:

DbMatchAny
(OR) Read all data for rows that are in *any* of the categories specified in the *pCategoryIDList* array.

DbMatchAll
(AND) Read all data for rows that are in *all* of the categories specified in the *pCategoryIDList* array and are possibly in others.

Schema Sync Manager API

SyncDbReadRows

DbMatchExact

Read all data for rows that are in *all* of the categories specified in the *pCategoryIDList* array and are in no others.

← *pNumRows*

A pointer to the number of rows in the *ppRows* array. This is the number of rows for which data is read or is available. Do not pass in NULL.

← *ppRows*

A pointer to the address of an array of **pNumRows* structures of type [DbRowData](#) that receives row data for matching rows. This function allocates memory for these structures, but the caller must call [SyncDbReleaseStorage\(\)](#) to release the allocated memory.

Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following error code values: SYNCERR_BAD_ARG One or more of the parameters passed in are invalid. SYNCERR_INVALID_CATEGORYID The allowed number of categories has been exceeded, or a category ID doesn't correspond to a defined category. SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value. SYNCERR_INVALID_MATCH_MODE The <i>matchMode</i> parameter specifies an invalid match mode. Specify only values described in " Category Match Modes " on page 30. SYNCERR_INVALID_TABLENAME The <i>pTableName</i> parameter specifies an invalid table name or a valid table name that does not exist. SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases. SYNCERR_ONE_OR_MORE_FAILED Retrieval of one or more rows' data failed. Check the <i>errCode</i> field of each DbRowData in the <i>ppRows</i> array for details.
----------------	--

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_TABLE_NAME_NOT_SPECIFIED

The *pTableName* parameter points to an empty character array.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

For a description of all Sync Manager error codes, see "[Schema Sync Manager Error Codes](#)" on page 207 and "[Common Sync Manager Error Codes](#)" on page 486.

Comments

This function retrieves the data set defined by *dbRowAllDataSet* in "[Row Data Sets](#)" on page 42 and allocates memory for the data for each row. When you no longer need to access the row data, you must pass the value of **ppRowInfoList* to [SyncDbReleaseStorage\(\)](#) to release the allocated memory.

Each [*DbRowData*](#).*dataSetRetrieved* field is set to *dbRowAllDataSet*, if the row is successfully retrieved. However, if this function returns SYNCERR_ONE_OR_MORE_FAILED, check the *errCode* field of each [*DbRowData*](#) structure in the *ppRows* array to determine which rows' values were successfully retrieved and which were not. For successfully retrieved row data, the *errCode* is SYNCERR_NONE; otherwise it is set to this error code:

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

If an error occurs for a row, that row's *RowData.errCode* field indicates the error and its *RowData.dataSetRetrieved* field indicates which data set was successfully retrieved. For example, if an error occurred and *dataSetRetrieved* is

(*dbRowInfoDataSet* | *dbRowCategoriesDataSet* | *dbRowColumnInfoDataSet*), then the error occurred while retrieving the data set defined by *dbRowColumnValuesDataSet*.

Schema Sync Manager API

SyncDbReadRows

Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	DbRowData , SyncDbReleaseStorage() , “ Row Data Sets ” on page 42

SyncDbReadRowsByRowInfo Function

Purpose Reads all of a row's data—the attributes, category membership, and all column values—for an array of [DbRowData](#) structures.

Declared In SyncDb.h

Prototype

```
HSError SyncDbReadRowsByRowInfo (HSByte handle,
                                  UInt32 numRows, DbRowData *pRowInfo,
                                  DbRowData **ppRows)
```

Parameters

→ *handle*
A handle to a schema database that is open for reading. This handle is returned by a call to [SyncDbOpenDatabase\(\)](#) or [SyncDbCreateDatabase\(\)](#).

→ *numRows*
The number of rows in the *pRowInfo* array.

→ *pRowInfo*
An array of *numRows* structures of type [DbRowData](#) that specifies at least the row IDs of the rows to read. You may specify the array passed back by a prior call to [SyncDbReadRowInfoList\(\)](#) or [SyncDbReadModifiedRowInfoList\(\)](#).

← *ppRows*
A pointer to the address of an array of **pNumRows* structures of type [DbRowData](#) that receives row data for the rows specified in the *pRowInfo* array. This function allocates memory for these structures, but the caller must call [SyncDbReleaseStorage\(\)](#) to release the allocated memory.

Returns If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following error code values:

SYNCERR_BAD_ARG
One or more of the parameters passed in are invalid.

SYNCERR_INVALID_FILE_HANDLE
The *handle* parameter is 0, which is an invalid value.

SYNCERR_NOT_SCHEMADB
The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

Schema Sync Manager API

SyncDbReadRowsByRowInfo

SYNCERR_ONE_OR_MORE_FAILED

Retrieval of one or more rows' data failed. Check the `errCode` field of each [DbRowData](#) in the `ppRows` array for details.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

For a description of all Sync Manager error codes, see "[Schema Sync Manager Error Codes](#)" on page 207 and "[Common Sync Manager Error Codes](#)" on page 486.

Comments

This function retrieves the data set defined by `dbRowAllDataSet` in "[Row Data Sets](#)" on page 42 and allocates memory for the data for each row. When you no longer need to access the row data, you must pass the value of `*ppRowInfoList` to [SyncDbReleaseStorage\(\)](#) to release the allocated memory.

Each `DbRowData`.`dataSetRetrieved` field is set to `dbRowAllDataSet`, if the row is successfully retrieved. However, if this function returns `SYNCERR_ONE_OR_MORE_FAILED`, check the `errCode` field of each `DbRowData` structure in the `ppRows` array to determine which rows values were successfully retrieved and which were not. For successfully retrieved row data, the `errCode` is `SYNCERR_NONE`; otherwise it is set to this error code:

SYNCERR_NOT_FOUND

The specified row is not present in the database.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

If an error occurs for a row, that row's `RowData`.`errCode` field indicates the error and its `RowData`.`dataSetRetrieved` field indicates which data set was successfully retrieved. For example, if an error occurred and `dataSetRetrieved` is `(dbRowInfoDataSet | dbRowCategoriesDataSet | dbRowColumnInfoDataSet)`, then the error occurred while retrieving the data set defined by `dbRowColumnValuesDataSet`.

A conduit that needs to synchronize a large amount of column data can use [SyncDbReadRowInfoList\(\)](#) or [SyncDbReadModifiedRowInfoList\(\)](#) together with this function to limit desktop memory usage. For example, a conduit may call [SyncDbReadModifiedRowInfoList\(\)](#) to identify all modified rows, then call [SyncDbReadRowsByRowInfo\(\)](#) multiple times to retrieve column values. This procedure allows each partial set of rows returned by [SyncDbReadRowsByRowInfo\(\)](#) to be freed before the next call.

Example	This example assumes that a database has 20,000 rows and processes them in batches of 1000 rows.
----------------	--

```
DbRowData* pRowInfo, prowData;
UInt32 numRows;
int i, batchSize = 1000;

SyncDbReadRowInfoList (handle, &numRows, &pRowInfo);

// Process rows in batches of 1000.
// This limits the amount of memory allocated at one time.

for(int i=0; i<numRows; i+=batchSize)
{
    SyncDbReadRowsByRowInfo (handle, batchSize,
        &(pRowInfo[i]), &prd);
    ProcessRows (handle, batchSize, prowData);
    SyncDbReleaseStorage (&prd);
}
SyncDbReleaseStorage (&pRowInfo);
```

Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
----------------------	---

See Also	DbRowData , SyncDbReleaseStorage() , “ Row Data Sets ” on page 42, SyncDbReadRowInfoList() , SyncDbReadModifiedRowInfoList()
-----------------	--

Schema Sync Manager API

SyncDbReleaseStorage

SyncDbReleaseStorage Function

Purpose Frees memory allocated by a call to a Sync Manager function that passes back a buffer.

Declared In SyncDb.h

Prototype HSError SyncDbReleaseStorage (void **pBuffer*)

Parameters → *pBuffer*

A pointer to the memory to be freed. This pointer must be passed back by a prior call to a function listed in the Comments section.

Returns If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following error code values:

SYNCERR_BAD_ARG

One or more of the parameters passed in are invalid.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

For a description of all Sync Manager error codes, see "[Schema Sync Manager Error Codes](#)" on page 207 and "[Common Sync Manager Error Codes](#)" on page 486.

Comments The following Sync Manager functions allocate memory and pass back a pointer to a buffer:

- [SyncDbReadRowInfo\(\)](#)
- [SyncDbReadRowInfoList\(\)](#)
- [SyncDbReadModifiedRowInfoList\(\)](#)
- [SyncDbReadRow\(\)](#)
- [SyncDbReadRows\(\)](#)
- [SyncDbReadModifiedRows\(\)](#)
- [SyncDbReadRowsByRowInfo\(\)](#)

After you are finished with the buffer allocated by any of these functions, you must call SyncDbReleaseStorage() to free it.

For example, if you called

```
SyncDbReadRows (dbHandle, NULL, 0, NULL, 0,  
&numRows, &pRows);
```

then you must call

```
SyncDbReleaseStorage (pRows);
```

to free the *pRows* buffer when you are done with it.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[SyncDbReadRowInfo\(\)](#), [SyncDbReadRowInfoList\(\)](#),
[SyncDbReadModifiedRowInfoList\(\)](#), [SyncDbReadRow\(\)](#),
[SyncDbReadRows\(\)](#), [SyncDbReadModifiedRows\(\)](#),
[SyncDbReadRowsByRowInfo\(\)](#)

Schema Sync Manager API

SyncDbRemoveCategory

SyncDbRemoveCategory Function

Purpose Removes a category from a schema database.

Declared In SyncDb.h

Prototype HSError SyncDbRemoveCategory (HSByte *handle*,
CategoryID *categoryID*)

Parameters

- *handle*
A handle to a schema database that is open for reading and writing. This handle is returned by [SyncDbOpenDatabase\(\)](#) or [SyncDbCreateDatabase\(\)](#).

- *categoryID*
The [CategoryID](#) value of a single category to remove from the database.

Returns If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following error code values:

SYNCERR_BAD_ARG

One or more of the parameters passed in are invalid.

SYNCERR_INVALID_CATEGORYID

The specified category ID doesn't correspond to a defined category.

SYNCERR_INVALID_FILE_HANDLE

The *handle* parameter is 0, which is an invalid value.

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

For a description of all Sync Manager error codes, see "[Schema Sync Manager Error Codes](#)" on page 207 and "[Common Sync Manager Error Codes](#)" on page 486.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [CategoryID](#), [SyncDbAddCategory\(\)](#)

SyncDbRemoveCategoryFromAllRows Function

Purpose	Removes category membership in the specified categories from all rows in the database, depending on the match mode criteria.
Declared In	<code>SyncDb.h</code>
Prototype	<pre>HSError SyncDbRemoveCategoryFromAllRows (HSByte handle, UInt32 numCategories, const CategoryID *pCategoryIDList, DbMatchModeType matchMode)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading and writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>numCategories</i> The number of categories in the <i>pCategoryIDList</i> array.</p> <p>→ <i>pCategoryIDList</i> An array of <i>numCategories</i> CategoryID values.</p> <p>→ <i>matchMode</i> One of the following DbMatchModeType values that specifies how the categories in the <i>pCategoryIDList</i> array must match a given row's category membership for this function to remove the category membership:</p> <p> DbMatchAny (OR) Remove categories from rows that are in <i>any</i> of the categories specified in the <i>pCategoryIDList</i> array.</p> <p> DbMatchAll (AND) Remove categories from rows that are in <i>all</i> of the categories specified in the <i>pCategoryIDList</i> array <i>and are possibly in others</i>.</p> <p> DbMatchExact Remove categories from rows that are in <i>all</i> of the categories specified in the <i>pCategoryIDList</i> array <i>and are in no others</i>.</p>

Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following error code values: SYNCERR_BAD_ARG One or more of the parameters passed in are invalid. SYNCERR_INVALID_CATEGORYID The allowed number of categories has been exceeded, or a category ID doesn't correspond to a defined category. SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value. SYNCERR_INVALID_MATCH_MODE The <i>matchMode</i> parameter specifies an invalid match mode. Specify only values described in " Category Match Modes " on page 30. SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases. SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld. SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below. For a description of all Sync Manager error codes, see " Schema Sync Manager Error Codes " on page 207 and " Common Sync Manager Error Codes " on page 486.
Comments	This function removes the specified category memberships from the rows that match the specified list of categories and match mode, but it does not remove the actual category definitions themselves, which are defined at the database level. The database must be opened with write access. The specified category IDs must be valid. This function might perform no action if either of the following is true: <ul style="list-style-type: none">• None of the supplied category IDs are valid and the match mode is DbMatchAny.

Schema Sync Manager API

SyncDbRemoveCategoryFromAllRows

- Any of the category IDs are not valid and the match mode is either DbMatchAll or DbMatchExact.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [CategoryID](#), “[Category Match Modes](#)” on page 30,
[SyncDbMoveRowsInCategory \(\)](#)

SyncDbRemoveColumnProperty Function

Purpose	Removes a single column property from a column in a database table.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbRemoveColumnProperty (HSByte handle, const char *tableName, ColumnID columnID, DbSchemaColumnProperty propID)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading and writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>tableName</i> The name of the table from which to remove column properties.</p> <p>→ <i>columnID</i> The ColumnID value of the column for which to remove a property.</p> <p>→ <i>propID</i> The DbSchemaColumnProperty value of the column property to remove.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_BUILT_IN_PROPERTY The column property ID specified in the <i>propID</i> parameter is that of a built-in property and cannot be removed.</p> <p>SYNCERR_INVALID_COLUMNID The column ID specified in the <i>columnID</i> parameter is invalid.</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_INVALID_PROPID The column property ID specified in the <i>propID</i> parameter is invalid.</p>

Schema Sync Manager API

SyncDbRemoveColumnProperty

SYNCERR_INVALID_TABLENAME

The *tableName* parameter specifies an invalid table name or a valid table name that does not exist.

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_TABLE_NAME_NOT_SPECIFIED

The *tableName* parameter specifies a null pointer or points to an empty character array.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

For a description of all Sync Manager error codes, see "[Schema Sync Manager Error Codes](#)" on page 207 and "[Common Sync Manager Error Codes](#)" on page 486.

Compatibility Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [ColumnID](#), [DbSchemaColumnProperty](#),
[SyncDbGetColumnPropertyValue\(\)](#),
[SyncDbSetColumnPropertyValue\(\)](#)

SyncDbRemoveColumns Function

Purpose	Removes one or more column definitions from a database schema and removes that column's data for all table rows described by that schema.
Declared In	<code>SyncDb.h</code>
Prototype	<pre>HSError SyncDbRemoveColumns (HSByte handle, const char *tableName, UInt32 numColumns, DbColumnRemove *pColumnRemoveList)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading and writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>tableName</i> The name of the table from which to remove columns.</p> <p>→ <i>numColumns</i> The number of columns in the <i>pColumnRemoveList</i> array.</p> <p>↔ <i>pColumnRemoveList</i> An array of <i>numColumns</i> structures of type DbColumnRemove that specifies the columns to remove to the table. Upon return, it receives a success or error code in the <i>errCode</i> field of each column that could not be removed.</p> <p>For a description of all Sync Manager error codes, see “Schema Sync Manager Error Codes” on page 207 and “Common Sync Manager Error Codes” on page 486.</p> <p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_INVALID_TABLENAME The <i>tableName</i> parameter specifies an invalid table name or a valid table name that does not exist.</p>

Schema Sync Manager API

SyncDbRemoveColumns

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_ONE_OR_MORE_FAILED

Removal of one or more columns failed. Check the *errCode* field of each [DbColumnRemove](#) structure in the *pColumnRemoveList* array for details.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_TABLE_NAME_NOT_SPECIFIED

The *tableName* parameter specifies a null pointer or points to an empty character array.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

If this function returns SYNCERR_ONE_OR_MORE_FAILED, check the *errCode* field of each [DbColumnRemove](#) structure in the *pColumnRemoveList* array to determine which columns were successfully removed and which were not. For successfully removed columns, the *errCode* is SYNCERR_NONE; otherwise it is set to the following error code:

SYNCERR_INVALID_COLUMNID

The column ID specified in *columnID* is invalid.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[DbColumnRemove](#), [SyncDbAddColumns\(\)](#)

SyncDbRemoveRow Function

Purpose	Removes a row specified by ID from a database and disposes of its data.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbRemoveRow (HSByte handle, RowID rowID)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading and writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>rowID</i> The RowID value identifying the row to remove.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_INVALID_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p>SYNCERR_NOT_FOUND The specified row is not in the database.</p> <p>SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.</p> <p>For a description of all Sync Manager error codes, see “Schema Sync Manager Error Codes” on page 207 and “Common Sync Manager Error Codes” on page 486.</p>
Comments	This function immediately disposes of the row’s data and removes the row’s entry in the database header.

Schema Sync Manager API

SyncDbRemoveRow

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [RowID](#), [SyncDbDeleteRow\(\)](#), [SyncDbCreateRow\(\)](#)

SyncDbRemoveRowCategory Function

Purpose	Removes membership in the specified categories from a single row.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbRemoveRowCategory (HSByte handle, RowID rowID, UInt32 numCategories, const CategoryID *pCategoryIDList)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading and writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>rowID</i> The RowID value identifying the row for which to remove category memberships.</p> <p>→ <i>numCategories</i> The number of categories in the <i>pCategoryIDList</i> array.</p> <p>→ <i>pCategoryIDList</i> An array of <i>numCategories</i> CategoryID values to remove from the row's category membership list. This function removes the row from all the categories on this list.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_CATEGORYID A specified category ID doesn't correspond to a defined category.</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p>

Schema Sync Manager API

SyncDbRemoveRowCategory

SYNCERR_NOT_FOUND

The specified row is not present in the database.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

This function removes the specified category memberships from the specified row but does not remove the actual category definitions themselves, which are defined at the database level.

The database must be opened with write access. The specified category IDs must be valid.

This function ignores category IDs for which the specified row is not a member. If the *pCategoryIDList* array contains multiple instances of a given category ID, the category membership is removed when the first instance is encountered; the remaining instances are ignored.

If *numCategories* is equal to one, and *pCategoryIDList[0]* is equal to *catIDAll*, this function removes all of the category memberships for this row.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[RowID](#), [CategoryID](#), [SyncDbAddRowCategory\(\)](#), “[Category ID Constants](#)” on page 30

SyncDbRemoveRows Function

Purpose Removes rows specified by a list of row IDs from a database and disposes of their data.

Declared In SyncDb.h

Prototype HSError SyncDbRemoveRows (HSByte *handle*,
 UInt32 *numRows*, DbRowResult **pRowRemove*)

Parameters

→ *handle*
A handle to a schema database that is open for reading and writing. This handle is returned by [SyncDbOpenDatabase\(\)](#) or [SyncDbCreateDatabase\(\)](#).

→ *numRows*
The number of rows in the *pRowRemove* array.

↔ *pRowRemove*
An array of *numRows* [DbRowResult](#) structures that specifies the row ID of each row to remove. Upon return, it receives a success or error code in each entry's *errCode* field.

Returns

If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following error code values:

SYNCERR_BAD_ARG
One or more of the parameters passed in are invalid.

SYNCERR_INVALID_FILE_HANDLE
The *handle* parameter is 0, which is an invalid value.

SYNCERR_NOT_SCHEMADB
The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_ONE_OR_MORE_FAILED
Removing one or more rows failed. Check the *errCode* field of each [DbRowResult](#) in the *pRowRemove* array for details.

SYNCERR_REMOTE_BAD_ARG
One or more invalid parameters has been passed to the handheld.

SYNCERR_UNKNOWN_REQUEST
The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

Schema Sync Manager API

SyncDbRemoveRows

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments	This function immediately disposes of the rows' data and removes the rows' entries in the database header. In the <i>pRowRemove</i> array, you must initialize all fields in each <i>DbRowResult</i> structure. Always initialize the <i>errCode</i> field to <i>SYNCERR_NONE</i> . If this function returns <i>SYNCERR_ONE_OR_MORE_FAILED</i> , check the <i>errCode</i> field of each <i>DbRowResult</i> structure in the <i>pRowRemove</i> array to determine which rows were successfully removed and which were not. For successfully removed rows, the <i>errCode</i> is <i>SYNCERR_NONE</i> ; otherwise it is set to one of these error codes: SYNCERR_NOT_FOUND The specified row is not in the database. SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	<i>DbRowResult</i> , <i>SyncDbRemoveRow()</i> , <i>SyncDbRemoveSecretRowsInTable()</i> , <i>SyncDbDeleteRows()</i> , <i>SyncDbCreateRows()</i>

SyncDbRemoveSecretRowsInTable Function

Purpose	Removes all private rows from a table and disposes of their data.
Declared In	<code>SyncDb.h</code>
Prototype	<code>HSError SyncDbRemoveSecretRowsInTable (HSByte handle, const char *tableName)</code>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading and writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>tableName</i> The name of the table from which to remove private rows. If you specify NULL, this function removes all private rows from all the tables in the database.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_INVALID_TABLENAME The <i>tableName</i> parameter specifies an invalid table name or a valid table name that does not exist.</p> <p>SYNCERR_NOT_FOUND The specified table is not present in the database.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p>SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.</p> <p>SYNCERR_TABLE_NAME_NOT_SPECIFIED The <i>tableName</i> parameter specifies a null pointer or points to an empty character array.</p>

Schema Sync Manager API

SyncDbRemoveSecretRowsInTable

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments This function immediately disposes of the private rows’ data and removes the private rows’ entries from the database header.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncDbRemoveRows \(\)](#),

SyncDbRemoveTable Function

Purpose	Removes a table definition from a schema database.
Declared In	<code>SyncDb.h</code>
Prototype	<code>HSError SyncDbRemoveTable (HSByte handle, const char *tableName)</code>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for reading and writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>tableName</i> The name of the table of which to remove the definition.</p>
Returns	<p>If successful, returns <code>SYNCERR_NONE</code>.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p><code>SYNCERR_ACCESS_DENIED</code> The Authorization Manager on the handheld denied access to the database, or the table contains one or more non-default sort indexes.</p> <p><code>SYNCERR_BAD_ARG</code> One or more of the parameters passed in are invalid.</p> <p><code>SYNCERR_INVALID_FILE_HANDLE</code> The <i>handle</i> parameter is 0, which is an invalid value.</p> <p><code>SYNCERR_INVALID_TABLENAME</code> The <i>tableName</i> parameter specifies an invalid table name or a valid table name that does not exist.</p> <p><code>SYNCERR_NOT_SCHEMADB</code> The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p><code>SYNCERR_REMOTE_BAD_ARG</code> One or more invalid parameters has been passed to the handheld.</p> <p><code>SYNCERR_ROWS_EXIST</code> The specified table contains non-deleted rows.</p> <p><code>SYNCERR_TABLE_NAME_NOT_SPECIFIED</code> The <i>tableName</i> parameter specifies a null pointer or points to an empty character array.</p>

Schema Sync Manager API

SyncDbRemoveTable

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

This function cannot remove a table if it contains one or more non-deleted rows or if any sort indexes are defined for the table. You must first delete or remove any such rows and sort indexes before you can remove the table. Because the schema Sync Manager API does not allow you to delete sort indexes, your application must delete them before a conduit can remove a table. If this function returns SYNCERR_ACCESS_DENIED, it may be because this table has nondefault sort indexes.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[SyncDbAddTable\(\)](#), [SyncDbGetTableSchema\(\)](#)

SyncDbSetColumnPropertyValue Function

Purpose	Sets a single property value for a table column.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbSetColumnPropertyValue (HSByte handle, const char *tableName, ColumnID columnID, DbSchemaColumnProperty propID, UInt32 numBytes, const HSByte *pPropertyValue)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>tableName</i> The name of the table for which to set a column property value.</p> <p>→ <i>columnID</i> The ColumnID value of the column for which to set a property value.</p> <p>→ <i>propID</i> The DbSchemaColumnProperty value of the column property for which to set a value.</p> <p>→ <i>numBytes</i> The number of bytes that the caller allocates in the <i>pPropertyValue</i> array.</p> <p>→ <i>pPropertyValue</i> An array of bytes that specifies the property value to set.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_BUILT_IN_PROPERTY The column property ID specified in the <i>propID</i> parameter is that of a built-in property and cannot be removed.</p> <p>SYNCERR_INVALID_COLUMNID The column ID specified in the <i>columnID</i> parameter is invalid.</p>

Schema Sync Manager API

SyncDbSetColumnPropertyValue

SYNCERR_INVALID_FILE_HANDLE

The *handle* parameter is 0, which is an invalid value.

SYNCERR_INVALID_PROPID

The column property ID specified in the *propID* parameter is invalid.

SYNCERR_INVALID_TABLENAME

The *tableName* parameter specifies an invalid table name or a valid table name that does not exist.

SYNCERR_NOT_SCHEMADB

The *handle* parameter refers to a nonschema database. This function operates only on schema databases.

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_TABLE_NAME_NOT_SPECIFIED

The *tableName* parameter specifies a null pointer or points to an empty character array.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

For a description of all Sync Manager error codes, see "[Schema Sync Manager Error Codes](#)" on page 207 and "[Common Sync Manager Error Codes](#)" on page 486.

Compatibility Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncDbGetColumnPropertyValue\(\)](#),
[SyncDbRemoveColumnProperty\(\)](#)

SyncDbSetRowCategory Function

Purpose	Sets category membership for a single database row.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbSetRowCategory (HSByte handle, RowID rowID, UInt32 numCategories, const CategoryID *pCategoryIDList)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>rowID</i> The RowID value identifying the row for which to set category memberships.</p> <p>→ <i>numCategories</i> The number of categories in the <i>pCategoryIDList</i> array.</p> <p>→ <i>pCategoryIDList</i> An array of <i>numCategories</i> CategoryID values to set as the row's category membership list. This function overwrites any existing category memberships with those on this list.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_CATEGORYID The allowed number of categories has been exceeded, or a category ID doesn't correspond to a defined category.</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p>

Schema Sync Manager API

SyncDbSetRowCategory

SYNCERR_NOT_FOUND

The specified row is not present in the database.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

Any previous category membership for the row is overwritten by the specified category membership. To remove all category membership from a row (making it “Unfiled”), set *numCategories* to 0 and *pCategoryIDList* to NULL.

The database must be opened with write access. The supplied category IDs must be valid.

If a given category ID occurs more than once in the category ID array, the row is made a member of the category and the duplicate category IDs are ignored.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[RowID](#), [CategoryID](#), [SyncDbAddCategory\(\)](#),
[SyncDbGetRowCategory\(\)](#), [SyncDbRemoveRowCategory\(\)](#)

SyncDbWriteColumnValue Function

Purpose	Writes the specified bytes of a single column value for a row.
Declared In	SyncDb.h
Prototype	<pre>HSError SyncDbWriteColumnValue (HSByte handle, RowID rowID, ColumnID columnID, UInt32 dataOffset, UInt32 dataSize, const HSByte *pData)</pre>
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>rowID</i> The RowID value identifying the row for which to write a column value.</p> <p>→ <i>columnID</i> The ColumnID value of the column for which to write a column value.</p> <p>→ <i>dataOffset</i> An offset from the first byte in the column value from which to start writing data.</p> <p>→ <i>dataSize</i> The number of bytes to write starting from the <i>dataOffset</i> position.</p> <p>→ <i>pData</i> A pointer to an array of HSByte values to write in the column value.</p>

Schema Sync Manager API

SyncDbWriteColumnValue

Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following error code values: SYNCERR_BAD_ARG One or more of the parameters passed in are invalid. SYNCERR_INVALID_COLUMNID The column ID specified in the <i>columnID</i> parameter is invalid. SYNCERR_INVALID_COLUMNSIZE The column value size specified in the <i>dataSize</i> parameter is invalid. SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value. SYNCERR_NOT_FOUND The specified row is not present in the database. SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases. SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld. SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.
Comments	To append data to the end of an existing column value, set the <i>dataOffset</i> parameter to kOffsetEndOfData. Using this special offset is equivalent to specifying an offset equal to the data size, but does not require a prior call to retrieve the size when it is not known in advance.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	RowID , ColumnID , SyncDbGetColumnValue() , SyncDbWriteColumnValues()

SyncDbWriteColumnValues Function

Purpose	Writes one or more column values for a row.
Declared In	SyncDb.h
Prototype	HSError SyncDbWriteColumnValues (HSByte <i>handle</i> , RowID <i>rowID</i> , UInt32 <i>numColumnValues</i> , DbColumnValue * <i>pColumnValues</i>)
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>rowID</i> The RowID value identifying the row for which to write column values.</p> <p>→ <i>numColumnValues</i> The number of column values in the <i>pColumnValues</i> array.</p> <p>↔ <i>pColumnValues</i> An array of <i>numColumnValues</i> structures of type DbColumnValue that specifies the columns values to write. Upon return, it receives a success or error code in each column value's <i>errCode</i> field.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_NOT_FOUND The specified row is not present in the database.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p>SYNCERR_OPERATION_ABORTED Writing column values aborted because of one or more invalid input parameters.</p>

Schema Sync Manager API

SyncDbWriteColumnValues

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

A NULL value for the `columnData` field of the [`DbColumnValue`](#) structure is allowed; this removes existing column data for the specified column and row.

If this function returns `SYNCERR_OPERATION_ABORTED`, check the `errCode` field of each [`DbColumnValue`](#) structure in the `pColumnValues` array to determine which rows were successfully written and which were not. For successfully written rows, the `errCode` is `SYNCERR_NONE`; otherwise it is set to one of these error codes:

SYNCERR_INVALID_COLUMNID

The column ID specified in `columnID` is invalid.

SYNCERR_INVALID_COLUMNSIZE

The column data size specified in `dataSize` is invalid.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[RowID](#), [`DbColumnValue`](#), [SyncDbGetColumnValues\(\)](#),
[SyncDbWriteColumnValue\(\)](#)

SyncDbWriteRow Function

Purpose Writes all of a row's data—the attributes, category membership, and all column values.

Declared In SyncDb.h

Prototype

```
HSError SyncDbWriteRow (HSByte handle,
    RowID rowID, UInt16 attr,
    UInt32 numCategories,
    const CategoryID *pCategoryIDList,
    UInt32 numColumnValues,
    DbColumnValue *pColumnValues)
```

Parameters

→ *handle*

A handle to a schema database that is open for writing. This handle is returned by [SyncDbOpenDatabase\(\)](#) or [SyncDbCreateDatabase\(\)](#).

→ *rowID*

The [RowID](#) value identifying the row to write.

→ *attr*

A combination of one or more nonexclusive attributes of this row. Specify values described in “[Row Attributes](#)” on page 40.

→ *numCategories*

The number of categories in the *pCategoryIDList* array.

→ *pCategoryIDList*

An array of *numCategories* [CategoryID](#) values to write to the row's category membership list. This function adds the row to all the categories on this list.

→ *numColumnValues*

The number of column values in the *pColumnValues* array.

↔ *pColumnValues*

An array of *numColumns* [DbColumnValue](#) structures that specify the column values to write. Upon return, it receives a success or error code in each column value's *errCode* field.

Schema Sync Manager API

SyncDbWriteRow

Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following error code values: SYNCERR_BAD_ARG One or more of the parameters passed in are invalid. SYNCERR_INVALID_CATEGORYID The allowed number of categories has been exceeded, or a category ID doesn't correspond to a defined category. SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value. SYNCERR_NOT_FOUND The specified row is not in the database. SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases. SYNCERR_ONE_OR_MORE_FAILED Writing one or more columns failed. Check the <i>errCode</i> field of each DbColumnValue in the <i>pColumnValues</i> array for details. SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld. SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.
Comments	You must create a row before you can write data in it; call SyncDbCreateRow() or SyncDbCreateRows() . You must initialize all fields in the <i>pColumnValues</i> array. In the <i>pCategoryIDList</i> field, never specify catIDAll. Always initialize the <i>errCode</i> field to SYNCERR_NONE. This function overwrites the category membership of the specified row. It updates only those column values that have been specified. If writing a column value fails, this function does not attempt to write the remaining column values; instead it sets the <i>errCode</i> field in

the failed [DbColumnValue](#) structures to SYNCERR_OPERATION_ABORTED.

To remove a column, pass in 0 for the dataSize field and NULL for the columnData field of the [DbColumnValue](#) structure that you want to remove.

If this function returns SYNCERR_ONE_OR_MORE_FAILED, check the errCode field of each [DbColumnValue](#) structure in the *pColumnValues* array to determine which column values were successfully written and which were not. For successfully written column values, the errCode is SYNCERR_NONE; otherwise it is set to one of these error codes:

SYNCERR_INVALID_COLUMNID

The column ID specified in columnID is invalid.

SYNCERR_INVALID_COLUMNSIZE

The column data size specified in dataSize is invalid.

SYNCERR_OPERATION_ABORTED

Writing column values aborted because writing the previous column value failed.

If *numCategories* is equal to one, and *pCategoryIDList[0]* is equal to *catIDUnfiled*, this function removes the category membership for this row.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[RowID](#), “[Row Attributes](#)” on page 40, [CategoryID](#), [DbColumnValue](#), [SyncDbCreateRow\(\)](#), [SyncDbCreateRows\(\)](#), [SyncDbReadRow\(\)](#), [SyncDbRemoveRow\(\)](#)

Schema Sync Manager API

SyncDbWriteRows

SyncDbWriteRows Function

Purpose	Writes all of the specified rows' data—the attributes, category membership, and all column values.
Declared In	SyncDb.h
Prototype	HSError SyncDbWriteRows (HSByte <i>handle</i> , UInt32 <i>numRows</i> , DbRowData * <i>prowData</i>)
Parameters	<p>→ <i>handle</i> A handle to a schema database that is open for writing. This handle is returned by SyncDbOpenDatabase() or SyncDbCreateDatabase().</p> <p>→ <i>numRows</i> The number of rows in the <i>prowData</i> array.</p> <p>↔ <i>prowData</i> An array of <i>numRows</i> DbRowData structures that specifies all the data to write for each row. Upon return, it receives a success or error code in each row's <i>errCode</i> field.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.</p> <p>SYNCERR_INVALID_FILE_HANDLE The <i>handle</i> parameter is 0, which is an invalid value.</p> <p>SYNCERR_NOT_SCHEMADB The <i>handle</i> parameter refers to a nonschema database. This function operates only on schema databases.</p> <p>SYNCERR_ONE_OR_MORE_FAILED Creating one or more rows failed. Check the <i>errCode</i> field of each DbRowData in the <i>prowData</i> array for details.</p> <p>SYNCERR_REMOTE_BAD_ARG One or more invalid parameters has been passed to the handheld.</p> <p>SYNCERR_STREAM_ERR An error occurred while streaming the data to the handheld.</p>

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

You must create a row before you can write data in it; call [SyncDbCreateRow \(\)](#) or [SyncDbCreateRows \(\)](#).

You must initialize all fields, except the *pTableName* field, in the *pRowData* array. In the *pCategoryIDList* field, never specify *catIDAll*. Always initialize the *errCode* field to *SYNCERR_NONE*. This function overwrites the category membership of the specified rows. It updates only those column values that have been specified. If writing fails for one row, this function continues to write the next row. If while writing a row, writing a column value fails, this function does not attempt to write the remaining column values; instead it sets the *errCode* field in the failed [DbColumnValue](#) structures to *SYNCERR_OPERATION_ABORTED*.

To remove a column in a **pColumnValues* array, pass in 0 for the *dataSize* field and *NULL* for the *columnData* field of the [DbColumnValue](#) structure that you want to remove.

If this function returns *SYNCERR_ONE_OR_MORE_FAILED*, check the *errCode* field of each [DbRowData](#) structure in the *pRowData* array to determine which rows were successfully written and which were not. For successfully written rows, the *errCode* is *SYNCERR_NONE*; otherwise it is set to one of these error codes:

SYNCERR_INVALID_TABLENAME

The table name specified in **pTableName* is invalid or does not exist.

SYNCERR_INVALID_CATEGORYID

The allowed number of categories has been exceeded, or a category ID doesn't correspond to a defined category.

SYNCERR_NOT_FOUND

The specified table is not in the database.

SYNCERR_ONE_OR_MORE_FAILED

Creating one or more rows failed. Check the *errCode* field of each [DbRowData](#) in the *pRowData* array for details.

Schema Sync Manager API

SyncDbWriteRows

SYNCERR_TABLE_NAME_NOT_SPECIFIED

The **pRowData.pTableName* field specifies an empty character array.

If the *errCode* field of a [DbRowData](#) structure in the *pRowData* array returns SYNCERR_ONE_OR_MORE_FAILED, check the *errCode* field of each [DbColumnValue](#) structure in the *pColumnValues* array to determine which column values were successfully written and which were not. For successfully written column values, the *errCode* is SYNCERR_NONE; otherwise it is set to one of these error codes:

SYNCERR_INVALID_COLUMNID

The column ID specified in *columnID* is invalid.

SYNCERR_INVALID_COLUMNSIZE

The column data size specified in *dataSize* is invalid.

SYNCERR_OPERATION_ABORTED

Writing column values aborted because writing the previous column value failed.

If *pRowData.numCategories* is equal to one, and *pRowData.pCategoryIDList[0]* is equal to *catIDUnfiled*, this function removes the category membership for this row.

Compatibility Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[DbRowData](#), [DbColumnValue](#), [SyncDbCreateRow\(\)](#),
[SyncDbCreateRows\(\)](#), [SyncDbReadRows\(\)](#),
[SyncDbRemoveRows\(\)](#)

SyncReadDatabaseList Function

Purpose	Retrieves information about a list of databases on the handheld.
Declared In	<code>SyncDb.h</code>
Prototype	<code>HSError SyncReadDatabaseList (UInt16 searchFlag, UInt32 *pCount, DBDatabaseInfo *pDatabaseInfo)</code>
Parameters	<p>→ <i>searchFlag</i> The kinds of databases to search for—for example, classic, extended, schema, secure, etc. Specify a combination of one or more of the values defined in “Database List Search Flags” on page 35.</p> <p>↔ <i>pCount</i> Specifies the number of elements that the caller allocates in the <i>pDatabaseInfo</i> array. The caller can pass in an address to a zero value, but it must not pass in <code>NULL</code>. Upon return, receives the number of databases for which this function retrieves information, if the number of elements in the <i>pDatabaseInfo</i> array is sufficient; or the number of matching databases on the handheld, if the number of elements in the <i>pDatabaseInfo</i> array is insufficient.</p> <p>← <i>pDatabaseInfo</i> An array of <i>*pCount</i> structures of type <code>DBDatabaseInfo</code> that receives database information in the section 1 fields of this structure. The caller can specify <code>NULL</code>, but only if <i>*pCount</i> is 0.</p>
Returns	<p>If successful, returns one of the following values:</p> <p><code>SYNCERR_MORE</code> The <i>pDatabaseInfo</i> array is not large enough. Allocate this array at least to the size passed back in <i>pCount</i> and call this function again to retrieve all database information.</p> <p><code>SYNCERR_NONE</code> The <i>pDatabaseInfo</i> array contains all the database information.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p><code>SYNCERR_BAD_ARG</code> One or more of the parameters passed in are invalid.</p>

Schema Sync Manager API

SyncReadDatabaseList

SYNCERR_REMOTE_BAD_ARG

One or more invalid parameters has been passed to the handheld.

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

For a description of all Sync Manager error codes, see “[Schema Sync Manager Error Codes](#)” on page 207 and “[Common Sync Manager Error Codes](#)” on page 486.

Comments

This function requires that the caller allocate memory for the *pDatabaseInfo* array of [DBDatabaseInfo](#) structures, as necessary.

Because you likely do not know the number of databases that can be returned in the list, you can call this function to retrieve the matching database count in **pCount*: pass in NULL in *pDatabaseInfo* and 0 in **pCount*. This function passes back in **pCount* the number of databases that match your search criteria and returns SYNCERR_MORE.

Now that you know the number of databases in the list, call this function again with a *pDatabaseInfo* array and **pCount* value equal to the size passed back from the first call.

The first time you call this function the Sync Manager retrieves the database information from the handheld and caches it on the desktop. The cache is preserved until you call the next Sync Manager function. If the call is not for the cached data, the cache is emptied. Therefore multiple calls to this function do not incur the performance penalty of retrieving data from the handheld each time over a possibly slow connection.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[DBDatabaseInfo](#), “[Database List Search Flags](#)” on page 35,
[SyncDbFindDatabase\(\)](#),
[SyncDbFindDatabaseByTypeCreator\(\)](#)

Schema Sync Manager Error Codes

[Table 2.1](#) lists the values of error codes that only schema Sync Manager functions can return. They can also return error codes described in “[Common Sync Manager Error Codes](#)” on page 486. The description of each function states which errors each function can return.

All of the named error codes below are defined as preprocessor constants, which are declared in the SyncDb.h header file.

Table 2.1 Schema Sync Manager error codes

Value	Error Code	Description
0x0004020	SYNCERR_MAX_CATEGORY_LIMIT	Adding another category exceeds the maximum number of categories allowed.
0x0004021	SYNCERR_NOT_SCHEMADB	Schema Sync API method used with a non-schema database.
0x0004022	SYNCERR_NOT_SECUREDB	Secure Sync API method used with a non-secure database.
0x0004023	SYNCERR_NOT_RECORDDB	Record Sync API method used with a resource database.
0x0004024	SYNCERR_INVALID_TABLENAME	An invalid table name or a valid table name that does not exist.
0x0004025	SYNCERR_INVALID_COLUMNID	The specified column ID is invalid.
0x0004026	SYNCERR_INVALID_PROPID	Invalid property ID specified in method.
0x0004027	SYNCERR_INVALID_CATEGORYID	A specified category ID does not correspond to a defined category.

Schema Sync Manager API

Schema Sync Manager Error Codes

Table 2.1 Schema Sync Manager error codes (*continued*)

Value	Error Code	Description
0x0004028	SYNCERR_INVALID_COLUMNTYPE	Invalid column data type specified in method.
0x0004029	SYNCERR_INVALID_COLUMNSPEC	Invalid column attributes specified in function.
0x000402A	SYNCERR_ONE_OR_MORE_FAILED	Retrieval, addition, or removal of one or more column definitions, column values, or column property values has failed.
0x000402B	SYNCERR_INVALID_TABLEDEFN	Invalid table definition.
0x000402C	SYNCERR_NO_DATA	There is no column data.
0x000402D	SYNCERR_ROWS_EXIST	Cannot remove the specified table because rows that belong to it still exist.
0x000402E	SYNCERR_COLUMNID_ALREADY_EXISTS	At least one specified column ID is defined more than once in a given table definition.
0x000402F	SYNCERR_BUILT_IN_PROPERTY	Cannot modify or remove a built-in column property.
0x0004030	SYNCERR_NAME_ALREADY_EXISTS	The specified category, table, or column name already exists.
0x0004031	SYNCERR_OPERATION_ABORTED	The writing of column values has aborted because of one or more invalid column values.
0x0004032	SYNCERR_INVALID_FILE_HANDLE	The specified database handle is 0, which is an invalid value.
0x0004033	SYNCERR_TABLE_NAME_NOT_SPECIFIED	A table name has not been specified.

Table 2.1 Schema Sync Manager error codes (*continued*)

Value	Error Code	Description
0x0004034	SYNCERR_CATEGORY_NAME_NOT_SPECIFIED	A category name has not been specified.
0x0004035	SYNCERR_INVALID_MATCH_MODE	An invalid match mode is specified.
0x0004036	SYNCERR_BACKUP_BIT_NOT_SET	Backup failed because the database's backup bit was not set.
0x0004037	SYNCERR_INVALID_ID	Invalid row ID specified when restoring a row.
0x0004038	SYNCERR_INVALID_COLUMNNAME	Invalid column name specified in method.
0x0004039	SYNCERR_INVALID_COLUMNSIZE	Invalid column size specified in create/write row method.
0x10004412	SYNCERR_REMOTE_PASSWORD	Backup or restore of security data failed because of an invalid password.
0x10006413	SYNCERR_STREAM_ERR	An error while streaming the data to the handheld.

Schema Sync Manager API

Schema Sync Manager Error Codes

Extended Sync Manager API

The extended Sync Manager API consists of only those functions that work with [extended databases](#). Other Sync Manager functions that do not depend on the database type are described in [Chapter 5, “Common Sync Manager API,”](#) on page 383.

This chapter has the following section:

[Extended Sync Manager Functions](#) 212

All Sync Manager functions are available in Sync20.dll. The API described in this chapter is declared in SyncDm.h.

See [Chapter 5, “Using the Extended Sync Manager API,”](#) on page 45 in the *C/C++ Sync Suite Companion* for more information.

Extended Sync Manager API

Extended Sync Manager Functions

Extended Sync Manager Functions

This section describes the following functions, which access only extended databases. See “[Common Sync Manager Functions and Macros](#)” on page 434 for functions that access all types of databases or utility functions and macros that do not access databases.

[SyncDmChangeCategory\(\)](#)

Moves all records from one category (the source category) to another category (the target category) in an extended database.

[SyncDmCloseDatabase\(\)](#)

Closes an extended database.

[SyncDmCloseDatabaseEx\(\)](#)

Closes an extended database and optionally updates its backup and modification dates.

[SyncDmCreateDatabase\(\)](#)

Creates a new [extended database](#) on the handheld, opens the database, and returns a handle to it.

[SyncDmDeleteAllResourceRecords\(\)](#)

Destroys all resources in an extended database on the handheld.

[SyncDmDeleteDatabase\(\)](#)

Deletes an extended database on the handheld.

[SyncDmDeleteRecord\(\)](#)

Removes a record from an extended database and disposes of its data.

[SyncDmDeleteResourceRecord\(\)](#)

Removes a resource from an extended database and disposes of its data.

[SyncDmFindDatabase\(\)](#)

Searches for an extended database by *name* and *creator ID* and returns information about the database, if it is found.

[SyncDmFindDatabaseByTypeCreator\(\)](#)

Searches for an extended database by *creator ID* and *type* and returns information about the database, if it is found.

SyncDmGetDatabaseRecordCount()	Retrieves the total record or resource count for an extended database on the handheld.
SyncDmMaxRemoteRecordSize()	Retrieves the maximum record or resource size supported by extended databases on the handheld.
SyncDmOpenDatabase()	Opens an existing extended database on the handheld.
SyncDmPurgeAllRecords()	Removes all the records from an extended database and disposes of their data, regardless of record status.
SyncDmPurgeAllRecordsInCategory()	Removes all the records in the specified category from an extended database and disposes of their data, regardless of record status.
SyncDmPurgeDeletedRecords()	Removes all the records marked as deleted or archived from an extended database and disposes of their data.
SyncDmReadAppInfoBlock()	Retrieves the specified bytes of the application info block, if one exists, from an extended database on the handheld.
SyncDmReadNextModifiedRecord()	Retrieves the specified bytes of the next record from an extended database on the handheld. Each call to this iterator function retrieves the next modified record—including deleted or archived records—until all modified records have been retrieved.
SyncDmReadNextModifiedRecordInCategory()	Retrieves the specified bytes of the next record in a category from an extended database on the handheld. Each call to this iterator function retrieves the next modified record in a category—including deleted or archived records—until all modified records have been retrieved.

Extended Sync Manager API

Extended Sync Manager Functions

[SyncDmReadNextRecordInCategory\(\)](#)

Retrieves the specified bytes of the next record in a category from an extended database on the handheld. Each call to this iterator function retrieves the next record in a category—regardless of whether it is deleted, archived, modified, or unchanged—until all records have been retrieved.

[SyncDmReadOpenDatabaseInfo\(\)](#)

Retrieves comprehensive information about an open, extended database on the handheld.

[SyncDmReadPositionXMap\(\)](#)

Retrieves a list of the record IDs in their sorted order from an extended database on the handheld.

[SyncDmReadRecordByID\(\)](#)

Retrieves the specified bytes of a record, by record ID, from an extended database on the handheld.

[SyncDmReadRecordByIndex\(\)](#)

Retrieves the specified bytes of a record, by index, from an extended database on the handheld.

[SyncDmReadResourceRecordByIndex\(\)](#)

Retrieves the specified bytes of a resource, by index, from an extended database on the handheld.

[SyncDmReadSortInfoBlock\(\)](#)

Retrieves the specified bytes of the sort info block, if one exists, from an extended database on the handheld.

[SyncDmResetRecordIndex\(\)](#)

Resets the record iteration index of an open, extended record database on the handheld.

[SyncDmResetSyncFlags\(\)](#)

Resets all records' modified flags and updates the backup date of an open, extended record database on the handheld.

[SyncDmWriteAppInfoBlock\(\)](#)

Writes the specified bytes of the application info block to an extended record or resource database on the handheld.

[SyncDmWriteRecord\(\)](#)

Writes the specified bytes of a record to an extended database on the handheld.

[SyncDmWriteResourceRecord\(\)](#)

Writes the specified bytes of a resource to an extended database on the handheld.

[SyncDmWriteSortInfoBlock\(\)](#)

Writes the specified bytes of the sort info block to an extended database on the handheld.

Extended Sync Manager API

SyncDmChangeCategory

SyncDmChangeCategory Function

Purpose	Moves all records from one category (the source category) to another category (the target category) in an extended database.
Declared In	SyncDm.h
Prototype	HSError SyncDmChangeCategory (HSByte <i>fHandle</i> , HSByte <i>from</i> , HSByte <i>to</i>)
Parameters	<p>→ <i>fHandle</i> A handle to an extended database on the handheld. This handle is returned by a call to SyncDmOpenDatabase() or SyncDmCreateDatabase(). The database must be opened for reading and writing.</p> <p>→ <i>from</i> The index of the source category. This must be a value between 0 and 15.</p> <p>→ <i>to</i> The index of the target category. This must be a value between 0 and 15.</p>
Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following nonzero error code values: SYNCERR_ARG_MISSING SYNCERR_BAD_ARG SYNCERR_BAD_OPERATION SYNCERR_COMM_NOT_INIT SYNCERR_LOCAL_CANCEL_SYNC SYNCERR_LOCAL_MEM SYNCERR_LOST_CONNECTION SYNCERR_NO_FILES_OPEN SYNCERR_READ_ONLY SYNCERR_REMOTE_BAD_ARG SYNCERR_REMOTE_MEM SYNCERR_REMOTE_SYS

SYNCERR_UNKNOWN

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

See "[Common Sync Manager Error Codes](#)" on page 486 for a description of all Sync Manager error codes.

Comments This function changes the category index of all records in a specified category in an open, extended database on the handheld, but it does not alter the modified status of the records.

The category index values specified by *from* and *to* must be in the range 0 to 15. The convention is that index 0 is the unfiled category and index values 1 through 15 are filed category index values.

Compatibility Sync Manager version: 2.4 or later.
Palm OS® version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncDmOpenDatabase\(\)](#), [SyncDmCreateDatabase\(\)](#)

Extended Sync Manager API

SyncDmCloseDatabase

SyncDmCloseDatabase Function

Purpose	Closes an extended database.
Declared In	SyncDm.h
Prototype	HSError SyncDmCloseDatabase (HSByte <i>fHandle</i>)
Parameters	<i>→ fHandle</i> A handle to an extended database on the handheld. This handle is returned by the call to SyncDmOpenDatabase() or SyncDmCreateDatabase() that opened this database.
Returns	If successful, returns SYNCERR_NONE, which means that the database was closed and its handle was destroyed. If unsuccessful, returns one of the following nonzero error code values: SYNCERR_ARG_MISSING SYNCERR_BAD_ARG SYNCERR_BAD_OPERATION SYNCERR_COMM_NOT_INIT SYNCERR_FILE_NOT_OPEN SYNCERR_LOCAL_CANCEL_SYNC SYNCERR_LOCAL_MEM SYNCERR_LOST_CONNECTION SYNCERR_NO_FILES_OPEN SYNCERR_NOT_FOUND SYNCERR_READ_ONLY SYNCERR_REMOTE_BAD_ARG SYNCERR_REMOTE_MEM SYNCERR_REMOTE_NO_SPACE SYNCERR_REMOTE_SYS SYNCERR_UNKNOWN SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments The Sync Manager allows multiple extended databases to be open at once, unlike classic databases. Therefore you need to close extended databases only before deleting them or before exiting your conduit.

If you also want to update an extended database’s backup and modification dates when you close the database, use [*SyncDmCloseDatabaseEx\(\)*](#) instead.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [*SyncDmOpenDatabase\(\)*](#), [*SyncDmCreateDatabase\(\)*](#),
[*SyncDmCloseDatabaseEx\(\)*](#)

Extended Sync Manager API

SyncDmCloseDatabaseEx

SyncDmCloseDatabaseEx Function

Purpose	Closes an extended database and optionally updates its backup and modification dates.
Declared In	SyncDm.h
Prototype	HSError SyncDmCloseDatabaseEx (HSByte <i>dbHandle</i> , HSByte <i>bOptFlags</i>)
Parameters	<p>→ <i>dbHandle</i> A handle to an extended database on the handheld. This handle is returned by the call to SyncDmOpenDatabase() or SyncDmCreateDatabase() that opened this database.</p> <p>→ <i>bOptFlags</i> Options for updating the database dates. You can combine the constants defined in “Database Closing Options” on page 410.</p>
Returns	If successful, returns SYNCERR_NONE, which means that the database was closed and its handle was destroyed. If unsuccessful, returns one of the following nonzero error code values: SYNCERR_ARG_MISSING SYNCERR_BAD_ARG SYNCERR_BAD_OPERATION SYNCERR_COMM_NOT_INIT SYNCERR_FILE_NOT_OPEN SYNCERR_LOCAL_CANCEL_SYNC SYNCERR_LOCAL_MEM SYNCERR_LOST_CONNECTION SYNCERR_NOT_FOUND SYNCERR_NO_FILES_OPEN SYNCERR_READ_ONLY SYNCERR_REMOTE_BAD_ARG SYNCERR_REMOTE_MEM SYNCERR_REMOTE_NO_SPACE

SYNCERR_REMOTE_SYS

SYNCERR_UNKNOWN

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

See "[Common Sync Manager Error Codes](#)" on page 486 for a description of all Sync Manager error codes.

Comments

This function is an extended version of [SyncDmCloseDatabase\(\)](#). SyncDmCloseDatabaseEx() takes an additional option flags parameter (*bOptFlags*) and can update the modification and backup dates of the database while closing it.

The Sync Manager allows multiple extended databases to be open at once, unlike classic databases. Therefore you need to close extended databases only before deleting them or before exiting your conduit.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[SyncDmOpenDatabase\(\)](#), [SyncDmCreateDatabase\(\)](#),
[SyncDmCloseDatabase\(\)](#)

Extended Sync Manager API

SyncDmCreateDatabase

SyncDmCreateDatabase Function

Purpose	Creates a new extended database on the handheld, opens the database, and returns a handle to it.
Declared In	SyncDm.h
Prototype	HSError SyncDmCreateDatabase (const char * <i>pName</i> , UInt32 <i>creator</i> , HSByte & <i>rHandle</i> , UInt32 <i>type</i> , eDbFlags <i>flags</i> , UInt16 <i>version</i>)
Parameters	<p>→ <i>pName</i> The database name.</p> <p>→ <i>creator</i> The database creator ID.</p> <p>← <i>rHandle</i> The handle to the database that was created.</p> <p>→ <i>type</i> The database type.</p> <p>→ <i>flags</i> An eDbFlags value that specifies several characteristics about the database to be created (whether it is to be backed up, read-only, and so on).</p> <p>→ <i>version</i> The version of the database. This value is application-specific and defaults to 0 if not specified.</p>
Returns	If successful, returns SYNCERR_NONE, which means that the database was created and its handle is passed back via <i>rHandle</i> . If unsuccessful, returns one of the following nonzero error code values: SYNCERR_ARG_MISSING SYNCERR_BAD_ARG SYNCERR_COMM_NOT_INIT SYNCERR_FILE_ALREADY_EXIST The specified database already exists on the handheld. SYNCERR_FILE_NOT_OPEN Unable to open the database once created. Should not occur in practice.

SYNCERR_LOCAL_CANCEL_SYNC
SYNCERR_LOCAL_MEM
SYNCERR_LOST_CONNECTION
SYNCERR_REMOTE_BAD_ARG
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_NO_SPACE
SYNCERR_REMOTE_SYS
SYNCERR_UNKNOWN

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments

After this function creates the extended database, it opens the database for exclusive read-write access, with private (secret) records shown.

Note that extended databases are uniquely identified by their database name and creator ID, unlike classic databases which are identified only by their name.

`SyncDmCreateDatabase()` cannot overwrite an existing database. If you attempt to create a database with the name and creator ID of an existing database, this function fails with the `SYNCERR_FILE_ALREADY_EXIST` error. To replace an existing extended database, explicitly delete the old database with [`SyncDmDeleteDatabase\(\)`](#) first and then call `SyncDmCreateDatabase()` to create the new extended database.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[`SyncDmDeleteDatabase\(\)`](#), [`SyncDmCloseDatabase\(\)`](#),
[`SyncDmCloseDatabaseEx\(\)`](#)

Extended Sync Manager API

SyncDmDeleteAllResourceRecords

SyncDmDeleteAllResourceRecords Function

Purpose	Destroys all resources in an extended database on the handheld.
Declared In	SyncDm.h
Prototype	HSError SyncDmDeleteAllResourceRecords (HSByte <i>fHandle</i>)
Parameters	→ <i>fHandle</i> A handle to an extended database on the handheld. This handle is returned by the call to SyncDmOpenDatabase() or SyncDmCreateDatabase() that opened this database.
Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following nonzero error code values: SYNCERR_ARG_MISSING SYNCERR_BAD_ARG SYNCERR_BAD_OPERATION SYNCERR_COMM_NOT_INIT SYNCERR_LOCAL_CANCEL_SYNC SYNCERR_LOCAL_MEM SYNCERR_LOST_CONNECTION SYNCERR_NO_FILES_OPEN SYNCERR_NOT_FOUND SYNCERR_READ_ONLY SYNCERR_REMOTE_BAD_ARG SYNCERR_REMOTE_CANCEL_SYNC SYNCERR_REMOTE_MEM SYNCERR_REMOTE_NO_SPACE SYNCERR_REMOTE_SYS SYNCERR_UNKNOWN SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments Before calling this function, the database must be open for reading and writing.

This function immediately disposes of all the resources' data chunks and removes the resources' entries from the database header. This is in contrast to some calls that applications on the handheld make to delete resources, which only mark those resources for deletion but do not destroy them immediately.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncDmOpenDatabase\(\)](#), [SyncDmCreateDatabase\(\)](#)

Extended Sync Manager API

SyncDmDeleteDatabase

SyncDmDeleteDatabase Function

Purpose	Deletes an extended database on the handheld.
Declared In	SyncDm.h
Prototype	<code>HSError SyncDmDeleteDatabase (const char *pName, UInt32 creator)</code>
Parameters	<p>→ <i>pName</i> The database name.</p> <p>→ <i>creator</i> The database creator ID.</p>
Returns	<p>If successful, returns SYNCERR_NONE, which means that the database was deleted.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>SYNCERR_ARG_MISSING SYNCERR_BAD_ARG SYNCERR_COMM_NOT_INIT SYNCERR_FILE_ALREADY_OPEN The database must be closed before it can be deleted.</p> <p>SYNCERR_LOCAL_CANCEL_SYNC SYNCERR_LOCAL_MEM SYNCERR_LOST_CONNECTION SYNCERR_NOT_FOUND SYNCERR_READ_ONLY The specified database is in ROM.</p> <p>SYNCERR_REMOTE_BAD_ARG SYNCERR_REMOTE_MEM SYNCERR_REMOTE_NO_SPACE SYNCERR_REMOTE_SYS SYNCERR_UNKNOWN</p>

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments You cannot delete an open database; you must close the database first.

Note that extended databases are uniquely identified by their database name and creator ID, unlike classic databases which are identified only by their name.

Compatibility Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncDmCloseDatabase\(\)](#), [SyncDmCloseDatabaseEx\(\)](#),
[SyncDmCreateDatabase\(\)](#)

Extended Sync Manager API

SyncDmDeleteRecord

SyncDmDeleteRecord Function

Purpose	Removes a record from an extended database and disposes of its data.
Declared In	SyncDm.h
Prototype	HSError SyncDmDeleteRecord (HSByte <i>handle</i> , UInt32 <i>recordID</i>)
Parameters	 → <i>handle</i> A handle to an extended database on the handheld. This handle is returned by the call to SyncDmOpenDatabase() or SyncDmCreateDatabase() that opened this database. → <i>recordID</i> The ID of the record to destroy.
Returns	If successful, returns SYNCERR_NONE, which means that the record was destroyed. If unsuccessful, returns one of the following nonzero error code values: SYNCERR_ARG_MISSING SYNCERR_BAD_ARG SYNCERR_BAD_OPERATION SYNCERR_COMM_NOT_INIT SYNCERR_LOCAL_CANCEL_SYNC SYNCERR_LOCAL_MEM SYNCERR_LOST_CONNECTION SYNCERR_NO_FILES_OPEN SYNCERR_NOT_FOUND SYNCERR_NOT_RECORDDB SYNCERR_READ_ONLY SYNCERR_REMOTE_BAD_ARG SYNCERR_REMOTE_MEM SYNCERR_REMOTE_SYS SYNCERR_UNKNOWN

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments The database must be open for reading and writing before a record can be deleted.

This function immediately disposes of the record’s data chunk and removes the record’s entry from the database header.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncDmOpenDatabase\(\)](#), [SyncDmCreateDatabase\(\)](#)

Extended Sync Manager API

SyncDmDeleteResourceRecord

SyncDmDeleteResourceRecord Function

Purpose	Removes a resource from an extended database and disposes of its data.
Declared In	SyncDm.h
Prototype	HSError SyncDmDeleteResourceRecord (HSByte <i>handle</i> , UInt32 <i>resourceType</i> , UInt16 <i>resourceID</i>)
Parameters	<p>→ <i>handle</i> A handle to an extended database on the handheld. This handle is returned by the call to SyncDmOpenDatabase() or SyncDmCreateDatabase() that opened this database.</p> <p>→ <i>resourceType</i> The type of the resource to delete.</p> <p>→ <i>resourceID</i> The ID of the resource to delete.</p>
Returns	If successful, returns SYNCERR_NONE, which means that the resource was destroyed. If unsuccessful, returns one of the following nonzero error code values: SYNCERR_ARG_MISSING SYNCERR_BAD_ARG SYNCERR_BAD_OPERATION SYNCERR_COMM_NOT_INIT SYNCERR_LOCAL_CANCEL_SYNC SYNCERR_LOCAL_MEM SYNCERR_LOST_CONNECTION SYNCERR_NO_FILES_OPEN SYNCERR_NOT_FOUND SYNCERR_READ_ONLY SYNCERR_REMOTE_BAD_ARG SYNCERR_REMOTE_MEM SYNCERR_REMOTE_NO_SPACE

SYNCERR_REMOTE_SYS

SYNCERR_UNKNOWN

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

See "[Common Sync Manager Error Codes](#)" on page 486 for a description of all Sync Manager error codes.

Comments The database must be open for reading and writing before a resource can be deleted.

This function immediately disposes of the resource's data chunk and removes the resource's entry from the database header.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncDmOpenDatabase\(\)](#), [SyncDmCreateDatabase\(\)](#)

Extended Sync Manager API

SyncDmFindDatabase

SyncDmFindDatabase Function

Purpose	Searches for an extended database by <i>name</i> and <i>creator ID</i> and returns information about the database, if it is found.
Declared In	SyncDm.h
Prototype	HSError SyncDmFindDatabase (const char * <i>pName</i> , UInt32 <i>creator</i> , HSByte <i>infoFlags</i> , DBDatabaseInfo * <i>pDatabaseInfo</i>)
Parameters	<p>→ <i>pName</i> The database name.</p> <p>→ <i>creator</i> The database creator ID.</p> <p>→ <i>infoFlags</i> Options to retrieve additional information about a matching database. Pass in a combination of one or more of the Database Information Retrieval Options (SYNC_DB_INFO_OPT_GET_...).</p> <p>← <i>pDatabaseInfo</i> A pointer to a DBDatabaseInfo structure that receives information about the matching database. The caller can pass in NULL.</p>
Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following nonzero error code values: SYNCERR_ARG_MISSING SYNCERR_BAD_ARG SYNCERR_COMM_NOT_INIT SYNCERR_LOCAL_CANCEL_SYNC SYNCERR_LOCAL_MEM SYNCERR_LOST_CONNECTION SYNCERR_NOT_FOUND The specified database was not found. SYNCERR_REMOTE_BAD_ARG SYNCERR_REMOTE_CANCEL_SYNC

SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_NO_SPACE
SYNCERR_REMOTE_SYS
SYNCERR_UNKNOWN
SYNCERR_UNKNOWN_REQUEST
The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.
See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments This function searches for an extended database only; it does not include classic or schema databases in its search.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [DBDatabaseInfo](#), [SyncDmFindDatabaseByTypeCreator\(\)](#)

Extended Sync Manager API

SyncDmFindDatabaseByTypeCreator

SyncDmFindDatabaseByTypeCreator Function

Purpose	Searches for an extended database by <i>creator ID</i> and <i>type</i> and returns information about the database, if it is found.
Declared In	SyncDm.h
Prototype	<pre>HSError SyncDmFindDatabaseByTypeCreator (UInt32 creator, UInt32 type, HSByte searchFlag, HSByte bOptFlags, DBDatabaseInfo *pDatabaseInfo)</pre>
Parameters	<p>→ <i>creator</i> The database creator ID.</p> <p>→ <i>type</i> The database type.</p> <p>→ <i>searchFlag</i> The search options for the find operation. This parameter determines whether this call searches for only the latest version of the database and whether the call is the start of a new search. Specify one or more of the values defined in “Database Search Options” on page 413 (SYNC_DB_SRCH_OPT_...). If you pass in zero, then this function continues the search for the next matching database, regardless of version.</p> <p>→ <i>bOptFlags</i> Specifies additional information to retrieve about a matching database. Pass in a combination of one or more of the values defined in “Database Information Retrieval Options” on page 412 (SYNC_DB_INFO_OPT_GET_...).</p> <p>← <i>pDatabaseInfo</i> A pointer to a DBDatabaseInfo structure that receives information about the matching database. The caller can pass in NULL.</p>
Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following nonzero error code values: SYNCERR_ARG_MISSING SYNCERR_BAD_ARG SYNCERR_COMM_NOT_INIT

SYNCERR_LOCAL_CANCEL_SYNC
SYNCERR_LOCAL_MEM
SYNCERR_LOST_CONNECTION
SYNCERR_NOT_FOUND
The specified database was not found.
SYNCERR_REMOTE_BAD_ARG
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_NO_SPACE
SYNCERR_REMOTE_SYS
SYNCERR_UNKNOWN
SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

See "[Common Sync Manager Error Codes](#)" on page 486 for a description of all Sync Manager error codes.

Comments This function can enumerate through multiple extended databases of a particular type or creator ID. To begin a new search for a specific creator ID/type pair, the caller must specify the `SYNC_DB_SRCH_OPT_NEW_SEARCH` flag in the *searchFlag* parameter. Subsequent calls in the same sequence must exclude this flag.

This function searches for an extended database only; it does not include classic or schema databases in its search.

Note that this function does not support creation or deletion of databases in the middle of enumerating through them.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [DBDatabaseInfo](#), [SyncDmFindDatabase\(\)](#)

Extended Sync Manager API

SyncDmGetDatabaseRecordCount

SyncDmGetDatabaseRecordCount Function

Purpose	Retrieves the total record or resource count for an extended database on the handheld.
Declared In	SyncDm.h
Prototype	HSError SyncDmGetDatabaseRecordCount (HSByte <i>fHandle</i> , UInt16 & <i>rNumRecs</i>)
Parameters	<p>→ <i>fHandle</i> A handle to an extended database on the handheld. This handle is returned by the call to SyncDmOpenDatabase() or SyncDmCreateDatabase() that opened this database.</p> <p>← <i>rNumRecs</i> The number of records or resources in the database.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>SYNCERR_ARG_MISSING SYNCERR_BAD_ARG SYNCERR_BAD_OPERATION SYNCERR_COMM_NOT_INIT SYNCERR_LOCAL_CANCEL_SYNC SYNCERR_LOCAL_MEM SYNCERR_LOST_CONNECTION SYNCERR_NO_FILES_OPEN The specified handle does not correspond to an open database on the handheld.</p> <p>SYNCERR_REMOTE_BAD_ARG The specified handle does not correspond to an open database on the handheld.</p> <p>SYNCERR_REMOTE_MEM SYNCERR_REMOTE_SYS SYNCERR_UNKNOWN</p>

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[SyncDmOpenDatabase\(\)](#), [SyncDmCreateDatabase\(\)](#)

Extended Sync Manager API

SyncDmMaxRemoteRecordSize

SyncDmMaxRemoteRecordSize Function

Purpose	Retrieves the maximum record or resource size supported by extended databases on the handheld.
Declared In	SyncDm.h
Prototype	HSError SyncDmMaxRemoteRecordSize (UInt32 &rdwMaxRecSize)
Parameters	$\leftarrow rdwMaxRecSize$ The maximum record size, in bytes, that can be allocated in an extended database on the handheld. This value is the maximum allowed size; the actual maximum size that can be allocated at a specific time is subject to available memory. Note the following special values: 0 Indicates that the maximum record size is unknown. 0xFFFFFFFF Indicates that there is no size limit, subject to available memory.
Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following nonzero error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below. See “ Common Sync Manager Error Codes ” on page 486 for a description of all Sync Manager error codes.
Comments	The maximum record size for extended databases is $2^{26} - 16$ bytes or ~64 MB, which is the value this function returns only for handhelds running Palm OS Cobalt.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.

SyncDmOpenDatabase Function

Purpose	Opens an existing extended database on the handheld.
Declared In	SyncDm.h
Prototype	<pre>HSError SyncDmOpenDatabase (const char *pName, UInt32 creator, HSByte &rHandle, HSByte openMode = (eDbWrite eDbRead eDbExclusive))</pre>
Parameters	<p>→ <i>pName</i> The database name.</p> <p>→ <i>creator</i> The database creator ID.</p> <p>← <i>rHandle</i> The handle to the database that was created.</p> <p>→ <i>openMode</i> A combination of one or more eDbOpenModes values to specify how to open the database—for example, open for reading and writing.</p>
Returns	<p>If successful, returns SYNCERR_NONE, which means that the database was opened.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>SYNCERR_ARG_MISSING SYNCERR_BAD_ARG SYNCERR_COMM_NOT_INIT SYNCERR_FILE_ALREADY_OPEN SYNCERR_LOCAL_CANCEL_SYNC SYNCERR_LOCAL_MEM SYNCERR_LOST_CONNECTION SYNCERR_NOT_FOUND There is no extended database on the handheld with the specified name and creator ID.</p> <p>SYNCERR_REMOTE_BAD_ARG SYNCERR_REMOTE_MEM</p>

Extended Sync Manager API

SyncDmOpenDatabase

SYNCERR_REMOTE_NO_SPACE

SYNCERR_REMOTE_SYS

SYNCERR_UNKNOWN

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments Sync Manager allows multiple *extended* databases to be open at once; therefore you need to close extended databases only before deleting them or before exiting your conduit.

Note that extended databases are uniquely identified by their database name and creator ID, unlike classic databases which are identified by their name and [memory card](#) number.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [eDbOpenModes](#), [SyncDmCloseDatabase\(\)](#),
[SyncDmCloseDatabaseEx\(\)](#), [SyncDmCreateDatabase\(\)](#)

SyncDmPurgeAllRecords Function

Purpose Removes all the records from an extended database and disposes of their data, regardless of record status.

Declared In SyncDm.h

Prototype HSError SyncDmPurgeAllRecords (HSByte *fHandle*)

Parameters → *fHandle*

A handle to an extended database on the handheld. This handle is returned by the call to [SyncDmOpenDatabase\(\)](#) or [SyncDmCreateDatabase\(\)](#) that opened this database.

Returns If successful, returns SYNCERR_NONE. Also returns SYNCERR_NONE if the database has no records.

If unsuccessful, returns one of the following nonzero error code values:

SYNCERR_ARG_MISSING
SYNCERR_BAD_ARG
SYNCERR_BAD_OPERATION
SYNCERR_COMM_NOT_INIT
SYNCERR_LOCAL_CANCEL_SYNC
SYNCERR_LOCAL_MEM
SYNCERR_LOST_CONNECTION
SYNCERR_NO_FILES_OPEN
SYNCERR_NOT_FOUND
SYNCERR_NOT_RECORDDB
SYNCERR_READ_ONLY
SYNCERR_REMOTE_BAD_ARG
SYNCERR_REMOTE_CANCEL_SYNC
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_SYS
SYNCERR_UNKNOWN

Extended Sync Manager API

SyncDmPurgeAllRecords

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments	This function immediately disposes of all the records’ data chunks and removes the records’ entries from the database header. The <i>fHandle</i> parameter must specify an extended database that is open for reading and writing.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncDmOpenDatabase () , SyncDmCreateDatabase () , SyncDmPurgeAllRecordsInCategory () , SyncDmPurgeDeletedRecords ()

SyncDmPurgeAllRecordsInCategory Function

Purpose Removes all the records in the specified category from an extended database and disposes of their data, regardless of record status.

Declared In SyncDm.h

Prototype HSError SyncDmPurgeAllRecordsInCategory
(HSByte *fHandle*, SInt16 *category*)

Parameters → *fHandle*
A handle to an extended database on the handheld. This handle is returned by the call to [SyncDmOpenDatabase\(\)](#) or [SyncDmCreateDatabase\(\)](#) that opened this database.

→ *category*
The index of the category whose records you want destroyed. By convention, use a value of 0 for the unfiled category, or use a value between 1 and 15 to specify a filed category.

Returns If successful, returns SYNCERR_NONE. Also returns SYNCERR_NONE if the database has no records.

If unsuccessful, returns one of the following nonzero error code values:

SYNCERR_ARG_MISSING
SYNCERR_BAD_ARG
SYNCERR_BAD_OPERATION
SYNCERR_COMM_NOT_INIT
SYNCERR_LOCAL_CANCEL_SYNC
SYNCERR_LOCAL_MEM
SYNCERR_LOST_CONNECTION
SYNCERR_NO_FILES_OPEN
SYNCERR_NOT_FOUND
SYNCERR_NOT_RECORDDB
SYNCERR_READ_ONLY
SYNCERR_REMOTE_BAD_ARG
SYNCERR_REMOTE_CANCEL_SYNC
SYNCERR_REMOTE_MEM

Extended Sync Manager API

SyncDmPurgeAllRecordsInCategory

SYNCERR_REMOTE_SYS

SYNCERR_UNKNOWN

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments For the specified database and category, this function immediately disposes of all the records’ data chunks and removes the records’ entries from the database header.

The *fHandle* parameter must specify an extended database that is open for reading and writing.

Note that this function does not update the record iteration index.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncDmOpenDatabase \(\)](#), [SyncDmCreateDatabase \(\)](#),
[SyncDmPurgeAllRecords \(\)](#),
[SyncDmPurgeDeletedRecords \(\)](#)

SyncDmPurgeDeletedRecords Function

Purpose Removes all the records marked as deleted or archived from an extended database and disposes of their data.

Declared In SyncDm.h

Prototype HSError SyncDmPurgeDeletedRecords (HSByte *fHandle*)

Parameters → *fHandle*

A handle to an extended database on the handheld. This handle is returned by the call to [SyncDmOpenDatabase\(\)](#) or [SyncDmCreateDatabase\(\)](#) that opened this database.

Returns If successful, returns SYNCERR_NONE. Also returns SYNCERR_NONE if the database has no records.

If unsuccessful, returns one of the following nonzero error code values:

SYNCERR_ARG_MISSING

SYNCERR_BAD_ARG

SYNCERR_BAD_OPERATION

SYNCERR_COMM_NOT_INIT

SYNCERR_LOCAL_CANCEL_SYNC

SYNCERR_LOCAL_MEM

SYNCERR_LOST_CONNECTION

SYNCERR_NO_FILES_OPEN

SYNCERR_NOT_FOUND

SYNCERR_NOT_RECORDDB

SYNCERR_READ_ONLY

SYNCERR_REMOTE_BAD_ARG

SYNCERR_REMOTE_CANCEL_SYNC

SYNCERR_REMOTE_MEM

SYNCERR_REMOTE_SYS

SYNCERR_UNKNOWN

Extended Sync Manager API

SyncDmPurgeDeletedRecords

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments

For all records whose delete bit is set, this function immediately disposes of all the records’ data chunks and removes the records’ entries from the database header.

The *fHandle* parameter must specify an extended database that is open for reading and writing.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[SyncDmOpenDatabase \(\)](#), [SyncDmCreateDatabase \(\)](#),
[SyncDmPurgeAllRecords \(\)](#),
[SyncDmPurgeAllRecordsInCategory \(\)](#)

SyncDmReadAppInfoBlock Function

Purpose Retrieves the specified bytes of the [application info](#) block, if one exists, from an extended database on the handheld.

Declared In SyncDm.h

Prototype

```
HSError SyncDmReadAppInfoBlock (HSByte handle,
                                UInt32 dataOffset, UInt32 *pDataSize,
                                HSBytePtr pAppInfoData,
                                UInt32 *pDataRemaining)
```

Parameters

→ *handle*

A handle to an extended database on the handheld. This handle is returned by the call to [SyncDmOpenDatabase\(\)](#) or [SyncDmCreateDatabase\(\)](#) that opened this database.

→ *dataOffset*

An offset from the first byte in the application info block from which to start retrieving data.

↔ *pDataSize*

Specifies the number of bytes of data to retrieve starting from the *dataOffset* position. The caller can pass in an address to a zero value, but it must not pass in NULL unless *pAppInfoData* is also NULL. Receives the number of bytes actually retrieved and stored in the *pAppInfoData* buffer.

← *pAppInfoData*

A pointer to an array of HSByte values that receives **pDataSize* bytes of the application info block data. The caller can pass in NULL.

← *pDataRemaining*

The number of bytes after the last one read (*dataOffset + *pDataSize*) to the end of the application info block. The caller can pass in NULL.

Returns If successful, returns one of the following values:

SYNCERR_NONE

The *pAppInfoData* array contains all of the application info block—that is, **pDataRemaining* = 0.

SYNCERR_MORE

The *pAppInfoData* array is not large enough to hold all of the data—that is, **pDataRemaining* > 0.

Extended Sync Manager API

SyncDmReadAppInfoBlock

If unsuccessful, returns one of the following negative error code values:

SYNCERR_ARG_MISSING
SYNCERR_BAD_ARG
SYNCERR_BAD_OPERATION
SYNCERR_COMM_NOT_INIT
SYNCERR_LOCAL_CANCEL_SYNC
SYNCERR_LOCAL_MEM
SYNCERR_LOST_CONNECTION
SYNCERR_NO_FILES_OPEN
SYNCERR_NOT_FOUND

The requested application info block cannot be found.

SYNCERR_REMOTE_BAD_ARG
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_NO_SPACE
SYNCERR_REMOTE_SYS
SYNCERR_UNKNOWN
SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments See the comments for [SyncDmReadRecordByID\(\)](#).

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncDmOpenDatabase\(\)](#), [SyncDmCreateDatabase\(\)](#),
[SyncDmWriteAppInfoBlock\(\)](#)

SyncDmReadNextModifiedRecord Function

Purpose	Retrieves the specified bytes of the next record from an extended database on the handheld. Each call to this iterator function retrieves the next modified record—including deleted or archived records—until all modified records have been retrieved.
Declared In	<code>SyncDm.h</code>
Prototype	<pre>HSError SyncDmReadNextModifiedRecord (HSByte handle, UInt16 *pRecordIndex, UInt32 *pRecordID, UInt16 *pCategoryIndex, HSByte *pAttributes, UInt32 dataOffset, UInt32 *pDataSize, HSBytePtr pRecordData, UInt32 *pDataRemaining)</pre>
Parameters	<p>→ <i>handle</i> A handle to an extended database on the handheld. This handle is returned by the call to SyncDmOpenDatabase() or SyncDmCreateDatabase() that opened this database.</p> <p>← <i>pRecordIndex</i> The index of the record. The caller can pass in NULL.</p> <p>← <i>pRecordID</i> The ID of the record. The caller can pass in NULL.</p> <p>← <i>pCategoryIndex</i> The index (0 to 15) of the category that the record belongs to. The caller can pass in NULL.</p> <p>← <i>pAttributes</i> The record attributes as a combination of one or more eSyncRecAttrs values. The caller can pass in NULL.</p> <p>→ <i>dataOffset</i> An offset from the first byte in the record data from which to start retrieving data.</p> <p>↔ <i>pDataSize</i> Specifies the number of bytes of data to retrieve starting from the <i>dataOffset</i> position. The caller can pass in an address to a zero value, but it must not pass in NULL unless <i>pRecordData</i> is also NULL. Receives the number of bytes actually retrieved and stored in the <i>pRecordData</i> buffer.</p>

Extended Sync Manager API

SyncDmReadNextModifiedRecord

$\leftarrow pRecordData$

A pointer to an array of HSByte values that receives $*pDataSize$ bytes of the record data. The caller can pass in NULL.

$\leftarrow pDataRemaining$

The number of bytes after the last one read ($dataOffset + *pDataSize$) to the end of the record data. The caller can pass in NULL.

Returns If successful, returns one of the following values:

SYNCERR_NONE

The $pRecordData$ array contains all of the record data—that is, $*pDataRemaining = 0$.

SYNCERR_MORE

The $pRecordData$ array is not large enough to hold all of the data—that is, $*pDataRemaining > 0$.

If unsuccessful, returns one of the following negative error code values:

SYNCERR_ARG_MISSING

SYNCERR_BAD_ARG

SYNCERR_BAD_OPERATION

SYNCERR_COMM_NOT_INIT

SYNCERR_LOCAL_CANCEL_SYNC

SYNCERR_LOCAL_MEM

SYNCERR_LOST_CONNECTION

SYNCERR_NO_FILES_OPEN

SYNCERR_NOT_FOUND

The requested record cannot be found.

SYNCERR_REMOTE_BAD_ARG

SYNCERR_REMOTE_CANCEL_SYNC

SYNCERR_REMOTE_MEM

SYNCERR_REMOTE_SYS

SYNCERR_UNKNOWN

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

See "[Common Sync Manager Error Codes](#)" on page 486 for a description of all Sync Manager error codes.

Comments

To read a portion of the record, the caller specifies an offset (*dataOffset*) from the first byte and the number of bytes to read (*pDataSize*). This function fills in the preallocated buffer with the specified number of bytes and sets **pDataSize* to the number of bytes it filled in. It also passes back the number of bytes remaining from *dataOffset + *pDataSize* to the end of the block.

The total record size is *dataOffset + *pDataSize + *pDataRemaining*.

NOTE: Each time you call this function, the Sync Manager advances the iteration pointer to the next record. Therefore, when this function returns SYNCERR_MORE indicating unread data is in the cache, call one of the non-iteration functions [*SyncDmReadRecordByID\(\)*](#) or [*SyncDmReadRecordByIndex\(\)*](#) to retrieve data from the cache without moving to the next record.

To retrieve all the data from a record of unknown size, the caller must perform the following actions:

1. Call this function to retrieve the size of the record. To do so, specify the following parameter values:
 - *handle* of the database you are iterating through
 - *dataOffset* = 0
 - **pDataSize* = 0
 - *pRecordData* = NULL
 - *pDataRemaining* as a pointer to a UInt32 to receive the size upon return

The Sync Manager retrieves the record from the handheld, caches it, passes back a pointer via *pDataRemaining* to the actual size of the record, and returns SYNCERR_MORE.

Extended Sync Manager API

SyncDmReadNextModifiedRecord

2. Allocate a buffer of the same number of HSByte values pointed to by the *pRecordData* value passed back in step 1.
3. Call either [*SyncDmReadRecordByID\(\)*](#) or [*SyncDmReadRecordByIndex\(\)*](#), specifying the following parameter values:
 - *handle* = the same value you specified in step 1
 - *recordID* or *pRecordIndex* = the value passed back by the call in step 1
 - *dataOffset* = the same value you specified in step 1
 - **pDataSize* = the value passed back in step 1
 - *pRecordData* = the same value passed back by the call in step 1.

The Sync Manager retrieves the data it cached in step 1, which is pointed to by *pRecordData* upon return. It also passes back the data size via *pDataSize*.

You can add, delete, or modify records between calls to this iterator function. You can also iterate through multiple extended database at the same time. This behavior differs from that of the classic Sync Manager iterator functions, which limit what other functions you can call during an iteration.

Compatibility Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [*eSyncRecAttrs*](#), [*SyncDmReadRecordByID\(\)*](#),
[*SyncDmReadRecordByIndex\(\)*](#), [*SyncDmResetRecordIndex\(\)*](#)

SyncDmReadNextModifiedRecordInCategory Function

Purpose	Retrieves the specified bytes of the next record in a category from an extended database on the handheld. Each call to this iterator function retrieves the next modified record in a category—including deleted or archived records—until all modified records have been retrieved.
Declared In	SyncDm.h
Prototype	<pre>HSError SyncDmReadNextModifiedRecordInCategory (HSByte handle, UInt16 *pRecordIndex, UInt32 *pRecordID, UInt16 categoryIndex, HSByte *pAttributes, UInt32 dataOffset, UInt32 *pDataSize, HSBytePtr pRecordData, UInt32 *pDataRemaining)</pre>
Parameters	<p>→ <i>handle</i> A handle to an extended database on the handheld. This handle is returned by the call to SyncDmOpenDatabase() or SyncDmCreateDatabase() that opened this database.</p> <p>← <i>pRecordIndex</i> The index of the record. The caller can pass in NULL.</p> <p>← <i>pRecordID</i> The ID of the record. The caller can pass in NULL.</p> <p>→ <i>categoryIndex</i> The index (0 to 15) of the category to iterate over.</p> <p>← <i>pAttributes</i> The record attributes as a combination of one or more eSyncRecAttrs values. The caller can pass in NULL.</p> <p>→ <i>dataOffset</i> An offset from the first byte in the record data from which to start retrieving data.</p> <p>↔ <i>pDataSize</i> Specifies the number of bytes of data to retrieve starting from the <i>dataOffset</i> position. The caller can pass in an address to a zero value, but it must not pass in NULL unless <i>pRecordData</i> is also NULL. Receives the number of bytes actually retrieved and stored in the <i>pRecordData</i> buffer.</p>

Extended Sync Manager API

SyncDmReadNextModifiedRecordInCategory

$\leftarrow pRecordData$

A pointer to an array of HSByte values that receives $*pDataSize$ bytes of the record data. The caller can pass in NULL.

$\leftarrow pDataRemaining$

The number of bytes after the last one read ($dataOffset + *pDataSize$) to the end of the record data. The caller can pass in NULL.

Returns If successful, returns one of the following values:

SYNCERR_NONE

The $pRecordData$ array contains all of the record data—that is, $*pDataRemaining = 0$.

SYNCERR_MORE

The $pRecordData$ array is not large enough to hold all of the data—that is, $*pDataRemaining > 0$.

If unsuccessful, returns one of the following negative error code values:

SYNCERR_ARG_MISSING

SYNCERR_BAD_ARG

SYNCERR_BAD_OPERATION

SYNCERR_COMM_NOT_INIT

SYNCERR_LOCAL_CANCEL_SYNC

SYNCERR_LOCAL_MEM

SYNCERR_LOST_CONNECTION

SYNCERR_NO_FILES_OPEN

SYNCERR_NOT_FOUND

The requested record cannot be found.

SYNCERR_REMOTE_BAD_ARG

SYNCERR_REMOTE_CANCEL_SYNC

SYNCERR_REMOTE_MEM

SYNCERR_REMOTE_SYS

SYNCERR_UNKNOWN

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

See "[Common Sync Manager Error Codes](#)" on page 486 for a description of all Sync Manager error codes.

Comments

See the comments for [SyncDmReadNextModifiedRecord\(\)](#).

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[eSyncRecAttrs](#), [SyncDmReadNextModifiedRecord\(\)](#),
[SyncDmReadRecordByID\(\)](#), [SyncDmReadRecordByIndex\(\)](#),
[SyncDmResetRecordIndex\(\)](#)

Extended Sync Manager API

SyncDmReadNextRecordInCategory

SyncDmReadNextRecordInCategory Function

Purpose	Retrieves the specified bytes of the next record in a category from an extended database on the handheld. Each call to this iterator function retrieves the next record in a category—regardless of whether it is deleted, archived, modified, or unchanged—until all records have been retrieved.
Declared In	SyncDm.h
Prototype	<pre>HSError SyncDmReadNextRecordInCategory (HSByte handle, UInt16 *pRecordIndex, UInt32 *pRecordID, UInt16 categoryIndex, HSByte *pAttributes, UInt32 dataOffset, UInt32 *pDataSize, HSBytePtr pRecordData, UInt32 *pDataRemaining)</pre>
Parameters	<p>→ <i>handle</i> A handle to an extended database on the handheld. This handle is returned by the call to SyncDmOpenDatabase() or SyncDmCreateDatabase() that opened this database.</p> <p>← <i>pRecordIndex</i> The index of the record. The caller can pass in NULL.</p> <p>← <i>pRecordID</i> The ID of the record. The caller can pass in NULL.</p> <p>→ <i>categoryIndex</i> The index (0 to 15) of the category to iterate over.</p> <p>← <i>pAttributes</i> The record attributes as a combination of one or more eSyncRecAttrs values. The caller can pass in NULL.</p> <p>→ <i>dataOffset</i> An offset from the first byte in the record data from which to start retrieving data.</p> <p>↔ <i>pDataSize</i> Specifies the number of bytes of data to retrieve starting from the <i>dataOffset</i> position. The caller can pass in an address to a zero value, but it must not pass in NULL unless <i>pRecordData</i> is also NULL. Receives the number of bytes actually retrieved and stored in the <i>pRecordData</i> buffer.</p>

$\leftarrow pRecordData$

A pointer to an array of HSByte values that receives $*pDataSize$ bytes of the record data. The caller can pass in NULL.

$\leftarrow pDataRemaining$

The number of bytes after the last one read ($dataOffset + *pDataSize$) to the end of the record data. The caller can pass in NULL.

Returns If successful, returns one of the following values:

SYNCERR_NONE

The $pRecordData$ array contains all of the record data—that is, $*pDataRemaining = 0$.

SYNCERR_MORE

The $pRecordData$ array is not large enough to hold all of the data—that is, $*pDataRemaining > 0$.

If unsuccessful, returns one of the following negative error code values:

SYNCERR_ARG_MISSING

SYNCERR_BAD_ARG

SYNCERR_BAD_OPERATION

SYNCERR_COMM_NOT_INIT

SYNCERR_LOCAL_CANCEL_SYNC

SYNCERR_LOCAL_MEM

SYNCERR_LOST_CONNECTION

SYNCERR_NO_FILES_OPEN

SYNCERR_NOT_FOUND

The requested record cannot be found.

SYNCERR_REMOTE_BAD_ARG

SYNCERR_REMOTE_CANCEL_SYNC

SYNCERR_REMOTE_MEM

SYNCERR_REMOTE_SYS

SYNCERR_UNKNOWN

Extended Sync Manager API

SyncDmReadNextRecordInCategory

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

- Comments** See the comments for [SyncDmReadNextModifiedRecord\(\)](#).
- Compatibility** Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.
- See Also** [eSyncRecAttrs](#), [SyncDmReadNextModifiedRecord\(\)](#),
[SyncDmReadRecordByID\(\)](#), [SyncDmResetRecordIndex\(\)](#)

SyncDmReadOpenDatabaseInfo Function

Purpose Retrieves comprehensive information about an open, extended database on the handheld.

Declared In SyncDm.h

Prototype

```
HSError SyncDmReadOpenDatabaseInfo (HSByte handle,  
                                    HSByte bOptFlags,  
                                    DBDatabaseInfo *pDatabaseInfo)
```

Parameters

→ *handle*
A handle to an extended database on the handheld. This handle is returned by the call to [SyncDmOpenDatabase\(\)](#) or [SyncDmCreateDatabase\(\)](#) that opened this database.

→ *bOptFlags*
Specifies additional information to retrieve about the database. Pass in a combination of one or more of the values defined in ["Database Information Retrieval Options"](#) on page 412 (SYNC_DB_INFO_OPT_GET_...).

← *pDatabaseInfo*
A pointer to a [DBDatabaseInfo](#) structure that receives information about the database. The caller can pass in NULL.

Returns If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following nonzero error code values:

SYNCERR_ARG_MISSING

SYNCERR_BAD_ARG

SYNCERR_BAD_OPERATION

SYNCERR_COMM_NOT_INIT

SYNCERR_LOCAL_CANCEL_SYNC

SYNCERR_LOCAL_MEM

SYNCERR_LOST_CONNECTION

SYNCERR_NO_FILES_OPEN

The specified handle does not correspond to an open database on the handheld.

Extended Sync Manager API

SyncDmReadOpenDatabaseInfo

SYNCERR_REMOTE_BAD_ARG

The specified handle does not correspond to an open handheld database.

SYNCERR_REMOTE_MEM

SYNCERR_REMOTE_SYS

SYNCERR_UNKNOWN

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments Ensure that the specified database is already open for reading before calling this function.about the database.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [DBDatabaseInfo](#), [SyncDmOpenDatabase\(\)](#),
[SyncDmCreateDatabase\(\)](#), [SyncDmFindDatabase\(\)](#),
[SyncDmFindDatabaseByTypeCreator\(\)](#)

SyncDmReadPositionXMap Function

Purpose Retrieves a list of the record IDs in their sorted order from an extended database on the handheld.

Declared In SyncDm.h

Prototype

```
HSError SyncDmReadPositionXMap (HSByte handle,
                                UInt16 startIndex, UInt16 dataSize,
                                HSBytePtr pRecordIDs, UInt16 *pNumReadIn)
```

Parameters

→ *handle*
A handle to an extended database on the handheld. This handle is returned by the call to [SyncDmOpenDatabase\(\)](#) or [SyncDmCreateDatabase\(\)](#) that opened this database.

→ *startIndex*
The index of the first record ID to read.

→ *dataSize*
The total number of bytes that you allocated for the *pRecordIDs* array. Because each record ID is four bytes long, specify a value that is four times the number of elements in the array.

← *pRecordIDs*
A pointer to an array of record IDs.

← *pNumReadIn*
The number of record IDs, not bytes, passed back in the *pRecordIDs* array.

Returns If successful, returns SYNCERR_NONE.
If unsuccessful, returns one of the following nonzero error code values:

SYNCERR_ARG_MISSING

SYNCERR_BAD_ARG

SYNCERR_BAD_OPERATION

SYNCERR_COMM_NOT_INIT

SYNCERR_FILE_NOT_OPEN

SYNCERR_LOCAL_CANCEL_SYNC

SYNCERR_LOCAL_MEM

Extended Sync Manager API

SyncDmReadPositionXMap

SYNCERR_LOST_CONNECTION
SYNCERR_NO_FILES_OPEN
SYNCERR_NOT_FOUND
SYNCERR_REMOTE_BAD_ARG
SYNCERR_REMOTE_CANCEL_SYNC
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_SYS
SYNCERR_UNKNOWN
SYNCERR_UNKNOWN_REQUEST
The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.
See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments	Conduits can use this function to apply the handheld’s record order to the database on the desktop computer. However, this approach might be slower than simply sorting the desktop records using the handheld sort criteria. This function always returns record ID values in little-endian (Intel) byte ordering format, unlike SyncReadPositionXMap() for classic databases.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncDmOpenDatabase() , SyncDmCreateDatabase()

SyncDmReadRecordByID Function

Purpose Retrieves the specified bytes of a record, by record ID, from an extended database on the handheld.

Declared In SyncDm.h

Prototype

```
HSError SyncDmReadRecordByID (HSByte handle,
    UInt16 *pRecordIndex, UInt32 recordID,
    UInt16 *pCategoryIndex, HSByte *pAttributes,
    UInt32 dataOffset, UInt32 *pDataSize,
    HSBytePtr pRecordData, UInt32 *pDataRemaining)
```

Parameters

→ *handle*
A handle to an extended database on the handheld. This handle is returned by the call to [SyncDmOpenDatabase\(\)](#) or [SyncDmCreateDatabase\(\)](#) that opened this database.

← *pRecordIndex*
The index of the record. The caller can pass in NULL.

→ *recordID*
The ID of the record to read.

← *pCategoryIndex*
The index (0 to 15) of the category that the record belongs to. The caller can pass in NULL.

← *pAttributes*
The record attributes as a combination of one or more [eSyncRecAttrs](#) values. The caller can pass in NULL.

→ *dataOffset*
An offset from the first byte in the record data from which to start retrieving data.

↔ *pDataSize*
Specifies the number of bytes of data to retrieve starting from the *dataOffset* position. The caller can pass in an address to a zero value, but it must not pass in NULL unless *pRecordData* is also NULL. Receives the number of bytes actually retrieved and stored in the *pRecordData* buffer.

← *pRecordData*
A pointer to an array of HSByte values that receives **pDataSize* bytes of the record data. The caller can pass in NULL.

Extended Sync Manager API

SyncDmReadRecordByID

$\leftarrow pDataRemaining$

The number of bytes after the last one read ($dataOffset + *pDataSize$) to the end of the record data. The caller can pass in NULL.

Returns If successful, returns one of the following values:

SYNCERR_NONE

The $pRecordData$ array contains all of the record data—that is, $*pDataRemaining = 0$.

SYNCERR_MORE

The $pRecordData$ array is not large enough to hold all of the data—that is, $*pDataRemaining > 0$.

If unsuccessful, returns one of the following negative error code values:

SYNCERR_ARG_MISSING

SYNCERR_BAD_ARG

SYNCERR_BAD_OPERATION

SYNCERR_COMM_NOT_INIT

SYNCERR_LOCAL_CANCEL_SYNC

SYNCERR_LOCAL_MEM

SYNCERR_LOST_CONNECTION

SYNCERR_NOT_FOUND

The requested record cannot be found.

SYNCERR_NO_FILES_OPEN

SYNCERR_REMOTE_BAD_ARG

SYNCERR_REMOTE_MEM

SYNCERR_REMOTE_SYS

SYNCERR_UNKNOWN

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments	<p>To read a portion of the record, the caller specifies an offset (<i>dataOffset</i>) from the first byte and the number of bytes to read (<i>pDataSize</i>). This function fills in the preallocated buffer with the specified number of bytes and sets <i>*pDataSize</i> to the number of bytes it filled in. It also passes back the number of bytes remaining from <i>dataOffset + pDataSize</i> to the end of the block.</p> <p>The total record size is <i>dataOffset + pDataSize + pDataRemaining</i>.</p> <p>To retrieve all the data from a record of unknown size, the caller must call this function two consecutive times: once to get the size of the record, so the caller can allocate memory for it, and again to fill in the buffer with data. However, multiple calls do not incur the performance penalty of retrieving data from the handheld each time over a possibly slow connection. Instead the Sync Manager retrieves data only once and caches it until either a consecutive call to this function returns SYNCERR_NONE or the caller calls a different Sync Manager function.</p> <p>To retrieve the size of the record, specify the following parameter values:</p> <ul style="list-style-type: none">• <i>dataOffset</i> = 0• <i>*pDataSize</i> = 0• <i>pRecordData</i> = NULL• <i>pDataRemaining</i> as a pointer to a UInt32 to receive the size upon return <p>Specify the database handle and then call this function. The Sync Manager retrieves the record from the handheld, caches it, and passes back a pointer via <i>pDataRemaining</i> to the actual size of the record.</p> <p>To retrieve all of the record, set <i>pDataSize</i> to at least the size passed back by the previous call. Allocate a buffer of the same number of HSByte values pointed to by <i>pRecordData</i>. Call this function again and specify the same database handle. Then the Sync Manager fills in the buffer with the record data and sets <i>pDataSize</i> to the number of bytes it fills in.</p>
-----------------	--

Extended Sync Manager API

SyncDmReadRecordByID

- Compatibility** Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.
- See Also** [eSyncRecAttrs](#), [SyncDmReadRecordByIndex\(\)](#)

SyncDmReadRecordByIndex Function

Purpose Retrieves the specified bytes of a record, by index, from an extended database on the handheld.

Declared In SyncDm.h

Prototype

```
HSError SyncDmReadRecordByIndex (HSByte handle,
                                 UInt16 recordIndex, UInt32 *pRecordID,
                                 UInt16 *pCategoryIndex, HSByte *pAttributes,
                                 UInt32 dataOffset, UInt32 *pDataSize,
                                 HSBytePtr pRecordData, UInt32 *pDataRemaining)
```

Parameters

→ *handle*
A handle to an extended database on the handheld. This handle is returned by the call to [SyncDmOpenDatabase\(\)](#) or [SyncDmCreateDatabase\(\)](#) that opened this database.

→ *recordIndex*
The index of the record to read.

← *pRecordID*
The ID of the record. The caller can pass in NULL.

← *pCategoryIndex*
The index (0 to 15) of the category that the record belongs to. The caller can pass in NULL.

← *pAttributes*
The record attributes as a combination of one or more [eSyncRecAttrs](#) values. The caller can pass in NULL.

→ *dataOffset*
An offset from the first byte in the record data from which to start retrieving data.

↔ *pDataSize*
Specifies the number of bytes of data to retrieve starting from the *dataOffset* position. The caller can pass in an address to a zero value, but it must not pass in NULL unless *pRecordData* is also NULL. Receives the number of bytes actually retrieved and stored in the *pRecordData* buffer.

← *pRecordData*
A pointer to an array of HSByte values that receives **pDataSize* bytes of the record data. The caller can pass in NULL.

Extended Sync Manager API

SyncDmReadRecordByIndex

$\leftarrow pDataRemaining$

The number of bytes after the last one read ($dataOffset + *pDataSize$) to the end of the record data. The caller can pass in NULL.

Returns If successful, returns one of the following values:

SYNCERR_NONE

The $pRecordData$ array contains all of the record data—that is, $*pDataRemaining = 0$.

SYNCERR_MORE

The $pRecordData$ array is not large enough to hold all of the data—that is, $*pDataRemaining > 0$.

If unsuccessful, returns one of the following negative error code values:

SYNCERR_ARG_MISSING

SYNCERR_BAD_ARG

SYNCERR_BAD_OPERATION

SYNCERR_COMM_NOT_INIT

SYNCERR_LOCAL_CANCEL_SYNC

SYNCERR_LOCAL_MEM

SYNCERR_LOST_CONNECTION

SYNCERR_NO_FILES_OPEN

SYNCERR_NOT_FOUND

The requested record cannot be found.

SYNCERR_REMOTE_BAD_ARG

SYNCERR_REMOTE_MEM

SYNCERR_REMOTE_SYS

SYNCERR_UNKNOWN

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

- Comments** See the comments for [SyncDmReadRecordByID\(\)](#).
- Compatibility** Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.
- See Also** [eSyncRecAttrs](#), [SyncDmReadRecordByID\(\)](#)

Extended Sync Manager API

SyncDmReadResourceRecordByIndex

SyncDmReadResourceRecordByIndex Function

Purpose Retrieves the specified bytes of a resource, by index, from an extended database on the handheld.

Declared In SyncDm.h

Prototype

```
HSError SyncDmReadResourceRecordByIndex
    (HSByte handle, UInt16 resourceIndex,
     UInt32 *pResourceType, UInt16 *pResourceID,
     UInt32 dataOffset, UInt32 *pDataSize,
     HSBytePtr pResourceData,
     UInt32 *pDataRemaining)
```

Parameters

→ *handle*
A handle to an extended database on the handheld. This handle is returned by the call to [SyncDmOpenDatabase\(\)](#) or [SyncDmCreateDatabase\(\)](#) that opened this database.

→ *resourceIndex*
The index of the resource to read.

← *pResourceType*
The type of the resource. The caller can pass in NULL.

← *pResourceID*
The ID of the resource. The caller can pass in NULL.

→ *dataOffset*
An offset from the first byte in the resource data from which to start retrieving data.

↔ *pDataSize*
Specifies the number of bytes of data to retrieve starting from the *dataOffset* position. The caller can pass in an address to a zero value, but it must not pass in NULL unless *pResourceData* is also NULL. Receives the number of bytes actually retrieved and stored in the *pResourceData* buffer.

← *pResourceData*
A pointer to an array of HSByte values that receives **pDataSize* bytes of the resource data. The caller can pass in NULL.

$\leftarrow pDataRemaining$

The number of bytes after the last one read ($dataOffset + *pDataSize$) to the end of the resource data. The caller can pass in NULL.

Returns If successful, returns one of the following values:

SYNCERR_NONE

The *pResourceData* array contains all of the resource data—that is, $*pDataRemaining = 0$.

SYNCERR_MORE

The *pResourceData* array is not large enough to hold all of the data—that is, $*pDataRemaining > 0$.

If unsuccessful, returns one of the following negative error code values:

SYNCERR_ARG_MISSING

SYNCERR_BAD_ARG

SYNCERR_BAD_OPERATION

SYNCERR_COMM_NOT_INIT

SYNCERR_LOCAL_CANCEL_SYNC

SYNCERR_LOCAL_MEM

SYNCERR_LOST_CONNECTION

SYNCERR_NO_FILES_OPEN

SYNCERR_NOT_FOUND

The requested resource cannot be found.

SYNCERR_REMOTE_BAD_ARG

SYNCERR_REMOTE_MEM

SYNCERR_REMOTE_SYS

SYNCERR_UNKNOWN

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Extended Sync Manager API

SyncDmReadResourceRecordByIndex

Comments See the comments for [SyncDmReadRecordByID\(\)](#).

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncDmReadRecordByID\(\)](#)

SyncDmReadSortInfoBlock Function

Purpose Retrieves the specified bytes of the sort info block, if one exists, from an extended database on the handheld.

Declared In SyncDm.h

Prototype

```
HSError SyncDmReadSortInfoBlock (HSByte handle,
                                 UInt32 dataOffset, UInt32 *pDataSize,
                                 HSBytePtr pSortInfoData,
                                 UInt32 *pDataRemaining)
```

Parameters

→ *handle*
A handle to an extended database on the handheld. This handle is returned by the call to [SyncDmOpenDatabase\(\)](#) or [SyncDmCreateDatabase\(\)](#) that opened this database.

→ *dataOffset*
An offset from the first byte in the sort info block from which to start retrieving data.

↔ *pDataSize*
Specifies the number of bytes of data to retrieve starting from the *dataOffset* position. The caller can pass in an address to a zero value, but it must not pass in NULL unless *pSortInfoData* is also NULL. Receives the number of bytes actually retrieved and stored in the *pSortInfoData* buffer.

← *pSortInfoData*
A pointer to an array of HSByte values that receives **pDataSize* bytes of the sort info block data. The caller can pass in NULL.

← *pDataRemaining*
The number of bytes after the last one read (*dataOffset + *pDataSize*) to the end of the sort info block. The caller can pass in NULL.

Returns If successful, returns one of the following values:

SYNCERR_NONE

The *pSortInfoData* array contains all of the sort info block—that is, **pDataRemaining* = 0.

SYNCERR_MORE

The *pSortInfoData* array is not large enough to hold all of the data—that is, **pDataRemaining* > 0.

Extended Sync Manager API

SyncDmReadSortInfoBlock

If unsuccessful, returns one of the following negative error code values:

SYNCERR_ARG_MISSING
SYNCERR_BAD_ARG
SYNCERR_BAD_OPERATION
SYNCERR_COMM_NOT_INIT
SYNCERR_LOCAL_CANCEL_SYNC
SYNCERR_LOCAL_MEM
SYNCERR_LOST_CONNECTION
SYNCERR_NO_FILES_OPEN
SYNCERR_NOT_FOUND

The requested sort info block cannot be found.

SYNCERR_REMOTE_BAD_ARG
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_SYS
SYNCERR_UNKNOWN
SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

See "[Common Sync Manager Error Codes](#)" on page 486 for a description of all Sync Manager error codes.

Comments	See the comments for SyncDmReadRecordByID() .
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncDmReadRecordByID() , SyncDmOpenDatabase() , SyncDmCreateDatabase() , SyncDmWriteSortInfoBlock()

SyncDmResetRecordIndex Function

Purpose Resets the record iteration index of an open, extended record database on the handheld.

Declared In SyncDm.h

Prototype HSError SyncDmResetRecordIndex (HSByte *fHandle*)

Parameters → *fHandle*

A handle to an extended database on the handheld. This handle is returned by the call to [SyncDmOpenDatabase\(\)](#) or [SyncDmCreateDatabase\(\)](#) that opened this database.

Returns If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following nonzero error code values:

SYNCERR_ARG_MISSING

SYNCERR_BAD_ARG

SYNCERR_BAD_OPERATION

SYNCERR_COMM_NOT_INIT

SYNCERR_LOCAL_CANCEL_SYNC

SYNCERR_LOCAL_MEM

SYNCERR_LOST_CONNECTION

SYNCERR_NO_FILES_OPEN

SYNCERR_REMOTE_BAD_ARG

SYNCERR_REMOTE_MEM

SYNCERR_UNKNOWN

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Extended Sync Manager API

SyncDmResetRecordIndex

Comments You do not need to call this function before iterating through a database for the first time; however, if your conduit iterates through the same database more than once while you have it open, you need to call this function to reset the index before beginning the second (or subsequent) iteration.

The extended Sync Manager API has the following iteration functions that are affected by this call:

- [SyncDmReadNextModifiedRecord\(\)](#)
- [SyncDmReadNextModifiedRecordInCategory\(\)](#)
- [SyncDmReadNextRecordInCategory\(\)](#)

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncDmReadNextModifiedRecord\(\)](#),
[SyncDmReadNextModifiedRecordInCategory\(\)](#),
[SyncDmReadNextRecordInCategory\(\)](#),
[SyncDmOpenDatabase\(\)](#), [SyncDmCreateDatabase\(\)](#)

SyncDmResetSyncFlags Function

Purpose Resets all records' modified flags and updates the backup date of an open, extended record database on the handheld.

Declared In SyncDm.h

Prototype HSError SyncDmResetSyncFlags (HSByte *fHandle*)

Parameters → *fHandle*

A handle to an extended database on the handheld. This handle is returned by the call to [SyncDmOpenDatabase\(\)](#) or [SyncDmCreateDatabase\(\)](#) that opened this database.

Returns If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following nonzero error code values:

SYNCERR_ARG_MISSING
SYNCERR_BAD_ARG
SYNCERR_BAD_OPERATION
SYNCERR_COMM_NOT_INIT
SYNCERR_LOCAL_CANCEL_SYNC
SYNCERR_LOCAL_MEM
SYNCERR_LOST_CONNECTION
SYNCERR_NO_FILES_OPEN
SYNCERR_NOT_FOUND
SYNCERR_NOT_RECORDDB
SYNCERR_READ_ONLY
SYNCERR_REMOTE_BAD_ARG
SYNCERR_REMOTE_CANCEL_SYNC
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_SYS
SYNCERR_UNKNOWN
SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

Extended Sync Manager API

SyncDmResetSyncFlags

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

- Comments** For a resource database, this function only updates the backup date. If *fHandle* is a handle to a record database, the database must be open for reading and writing. If *fHandle* is a handle to a resource database, the database can be open for either read-only or read-write access. For more information about the record flags, see “[eSyncRecAttrs](#)” on page 425.

NOTE: `SyncDmResetSyncFlags()` works only for extended databases. To reset the modification state for a schema database, see the description of [SyncDbCloseDatabase\(\)](#) instead.

- Compatibility** Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

- See Also** [eSyncRecAttrs](#), [SyncDmOpenDatabase\(\)](#),
[SyncDmCreateDatabase\(\)](#)

SyncDmWriteAppInfoBlock Function

Purpose Writes the specified bytes of the [application info](#) block to an extended record or resource database on the handheld.

Declared In SyncDm.h

Prototype

```
HSError SyncDmWriteAppInfoBlock (HSByte handle,
                                 UInt32 dataOffset, UInt32 dataSize,
                                 HSBytePtr pAppInfoBlock)
```

Parameters

- *handle*
A handle to an extended database on the handheld. This handle is returned by the call to [SyncDmOpenDatabase\(\)](#) or [SyncDmCreateDatabase\(\)](#) that opened this database.
- *dataOffset*
An offset from the first byte in the application info block from which to start writing data.
- *dataSize*
The number of bytes of data to write starting from the *dataOffset* position. The caller can specify 0.
- *pAppInfoBlock*
A pointer to an array of HSByte values that specifies *dataSize* bytes to write to the application info block. The caller can specify NULL.

Returns If successful, returns SYNCERR_NONE, which means that the block was written to the handheld database.

If unsuccessful, returns one of the following nonzero error code values:

- SYNCERR_ARG_MISSING
- SYNCERR_BAD_ARG
- SYNCERR_BAD_OPERATION
- SYNCERR_COMM_NOT_INIT
- SYNCERR_LIMIT_EXCEEDED
- SYNCERR_LOCAL_CANCEL_SYNC
- SYNCERR_LOCAL_MEM
- SYNCERR_LOST_CONNECTION

Extended Sync Manager API

SyncDmWriteAppInfoBlock

SYNCERR_NO_FILES_OPEN
SYNCERR_READ_ONLY
SYNCERR_REMOTE_BAD_ARG
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_NO_SPACE
SYNCERR_REMOTE_SYS
SYNCERR_UNKNOWN
SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

See "[Common Sync Manager Error Codes](#)" on page 486 for a description of all Sync Manager error codes.

Comments	<p>The database must be open for writing before calling this function.</p> <p>To write the application info block, specify an offset (<i>dataOffset</i>) from the first byte of the application info block and the number of bytes (<i>dataSize</i>) to write. This function writes <i>dataSize</i> bytes to the application info block from the buffer pointed to by <i>pAppInfoBlock</i>. The total application info block size after a successful write is <i>dataOffset</i> + <i>dataSize</i>. Any existing data following the <i>dataOffset</i> + <i>dataSize</i> position is discarded.</p> <p>To truncate the application info block at a given position, specify <i>dataSize</i> = 0 and <i>dataOffset</i> > 0 (the position from which to truncate). The <i>dataOffset</i> value you specify becomes the new application info block size.</p> <p>To delete the application info block, specify zero for the <i>dataSize</i>, <i>dataOffset</i>, and <i>pAppInfoBlock</i> parameters.</p> <p>To append data to the end of existing application info block data, set the <i>dataOffset</i> parameter to kOffsetEndOfData. Using this special offset is equivalent to specifying an offset equal to the data size, but does not require a prior call to retrieve the size when it is not known in advance.</p>
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncDmReadAppInfoBlock()

SyncDmWriteRecord Function

Purpose Writes the specified bytes of a record to an extended database on the handheld.

Declared In SyncDm.h

Prototype

```
HSError SyncDmWriteRecord (HSByte handle,
                           UInt32 *pRecordID, UInt16 categoryIndex,
                           HSByte attributes, UInt32 dataOffset,
                           UInt32 dataSize, HSBytePtr pRecordData)
```

Parameters

→ *handle*
A handle to an extended database on the handheld. This handle is returned by the call to [SyncDmOpenDatabase\(\)](#) or [SyncDmCreateDatabase\(\)](#) that opened this database.

↔ *pRecordID*
The ID of the record to write. To update or restore an existing record, specify the unique, nonzero record ID it was previously assigned on the handheld. To add a new record, specify a record ID of 0; the handheld assigns a unique record ID and passes it back via this parameter.

→ *categoryIndex*
The index (0 to 15) of the category that the record belongs to.

→ *attributes*
The record attributes as a combination of any of the following [eSyncRecAttrs](#) values: eRecAttrDeleted, eRecAttrDirty, eRecAttrSecret, eRecAttrArchived, or none.

→ *dataOffset*
An offset from the first byte in the record data from which to start writing data.

→ *dataSize*
The number of bytes of data to write starting from the *dataOffset* position. If you pass in 0, *attributes* must include eRecAttrDeleted.

→ *pRecordData*
A pointer to an array of HSByte values that specifies *dataSize* bytes to write to the record.

Extended Sync Manager API

SyncDmWriteRecord

Returns If successful, returns SYNCERR_NONE, which means that the record was written.

If unsuccessful, returns one of the following nonzero error code values:

SYNCERR_ARG_MISSING
SYNCERR_BAD_ARG
SYNCERR_BAD_OPERATION
SYNCERR_COMM_NOT_INIT
SYNCERR_LIMIT_EXCEEDED
SYNCERR_LOCAL_CANCEL_SYNC
SYNCERR_LOCAL_MEM
SYNCERR_LOST_CONNECTION
SYNCERR_NO_FILES_OPEN
SYNCERR_NOT_FOUND
SYNCERR_NOT_RECORDDB
SYNCERR_READ_ONLY
SYNCERR_REMOTE_BAD_ARG
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_NO_SPACE
SYNCERR_REMOTE_SYS
SYNCERR_UNKNOWN
SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments This function can perform the following write operations on an extended database that is open for writing:

- Overwrite, add to, truncate, or delete data in an existing record by specifying its record ID.
- Add a new record by supplying 0 as the record ID. Note that the Data Manager on the handheld is responsible for assigning new record IDs.

You cannot specify the position in the database of a new record.

Upon completion of synchronization operations, applications on the handheld are sent a *sysAppLaunchCmdSyncNotify* notification, at which time an application can sort or update the database as required.

To write a record, specify an offset (*dataOffset*) from the first byte of the record and the number of bytes (*dataSize*) to write. This function writes *dataSize* bytes to the record from the buffer pointed to by *pRecordData*. The total record size after a successful write is *dataOffset* + *dataSize*. Any existing data following the *dataOffset* + *dataSize* position is discarded.

To truncate a record at a given position, specify *dataSize* = 0 and *dataOffset* > 0 (the position from which to truncate). The *dataOffset* value you specify becomes the new record size.

To delete all of the record's data, specify zero for the *dataSize* and *dataOffset* parameters, NULL for the *pRecordData* parameter, and include *eRecAttrDeleted* in the *attributes* parameter.

To append data to the end of existing record data, set the *dataOffset* parameter to *kOffsetEndOfData*. Using this special offset is equivalent to specifying an offset equal to the data size, but does not require a prior call to retrieve the size when it is not known in advance.

Compatibility Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [*eSyncRecAttrs*](#), [*SyncDmReadRecordByID\(\)*](#),
[*SyncDmReadRecordByIndex\(\)*](#)

Extended Sync Manager API

SyncDmWriteResourceRecord

SyncDmWriteResourceRecord Function

Purpose Writes the specified bytes of a resource to an extended database on the handheld.

Declared In SyncDm.h

Prototype HSError SyncDmWriteResourceRecord (HSByte handle,
 UInt32 resourceType, UInt16 resourceId,
 UInt32 dataOffset, UInt32 dataSize,
 HSBytePtr pResourceData)

Parameters

- *handle*
A handle to an extended database on the handheld. This handle is returned by the call to [SyncDmOpenDatabase\(\)](#) or [SyncDmCreateDatabase\(\)](#) that opened this database.
- *resourceType*
The type of the resource.
- *resourceID*
The ID of the resource.
- *dataOffset*
An offset from the first byte in the resource data from which to start writing data.
- *dataSize*
The number of bytes of data to write starting from the *dataOffset* position.
- *pResourceData*
A pointer to an array of HSByte values that specifies *dataSize* bytes to write to the resource.

Returns If successful, returns SYNCERR_NONE, which means that the resource was written.

If unsuccessful, returns one of the following nonzero error code values:

- SYNCERR_ARG_MISSING
- SYNCERR_BAD_ARG
- SYNCERR_BAD_OPERATION
- SYNCERR_COMM_NOT_INIT
- SYNCERR_LIMIT_EXCEEDED

SYNCERR_LOCAL_CANCEL_SYNC
SYNCERR_LOCAL_MEM
SYNCERR_LOST_CONNECTION
SYNCERR_NO_FILES_OPEN
SYNCERR_NOT_FOUND
SYNCERR_READ_ONLY
SYNCERR_REMOTE_BAD_ARG
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_NO_SPACE
SYNCERR_REMOTE_SYS
SYNCERR_UNKNOWN
SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

See "[Common Sync Manager Error Codes](#)" on page 486 for a description of all Sync Manager error codes.

Comments

The database must be open for writing before calling this function.

To write a resource, specify an offset (*dataOffset*) from the first byte of the resource and the number of bytes (*dataSize*) to write. This function writes *dataSize* bytes to the resource from the buffer pointed to by *pResourceData*. The total resource size after a successful write is *dataOffset* + *dataSize*. Any existing data following the *dataOffset* + *dataSize* position is discarded.

To truncate a resource at a given position, specify *dataSize* = 0 and *dataOffset* > 0 (the position from which to truncate). The *dataOffset* value you specify becomes the new resource size.

To delete all of the resource's data, specify zero for the *dataSize* and *dataOffset* parameters, and NULL for the *pResourceData* parameter.

To append data to the end of existing resource data, set the *dataOffset* parameter to kOffsetEndOfData. Using this special offset is equivalent to specifying an offset equal to the data size, but

Extended Sync Manager API

SyncDmWriteResourceRecord

does not require a prior call to retrieve the size when it is not known in advance.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncDmReadResourceRecordByIndex\(\)](#)

SyncDmWriteSortInfoBlock Function

Purpose Writes the specified bytes of the sort info block to an extended database on the handheld.

Declared In SyncDm.h

Prototype

```
HSError SyncDmWriteSortInfoBlock (HSByte handle,
                                  UInt32 dataOffset, UInt32 dataSize,
                                  HSBytePtr pSortInfoBlock)
```

Parameters

- *handle*
A handle to an extended database on the handheld. This handle is returned by the call to [SyncDmOpenDatabase\(\)](#) or [SyncDmCreateDatabase\(\)](#) that opened this database.
- *dataOffset*
An offset from the first byte in the sort info block from which to start writing data.
- *dataSize*
The number of bytes of data to write starting from the *dataOffset* position.
- *pSortInfoBlock*
A pointer to an array of HSByte values that specifies *dataSize* bytes to write to the sort info block.

Returns If successful, returns SYNCERR_NONE, which means the block was written to the handheld database.

If unsuccessful, returns one of the following nonzero error code values:

- SYNCERR_ARG_MISSING
- SYNCERR_BAD_ARG
- SYNCERR_BAD_OPERATION
- SYNCERR_COMM_NOT_INIT
- SYNCERR_LIMIT_EXCEEDED
- SYNCERR_LOCAL_CANCEL_SYNC
- SYNCERR_LOCAL_MEM
- SYNCERR_LOST_CONNECTION
- SYNCERR_NO_FILES_OPEN

Extended Sync Manager API

SyncDmWriteSortInfoBlock

SYNCERR_READ_ONLY
SYNCERR_REMOTE_BAD_ARG
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_NO_SPACE
SYNCERR_REMOTE_SYS
SYNCERR_UNKNOWN
SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments	<p>The database must be open for writing before calling this function.</p> <p>To write the sort info block, specify an offset (<i>dataOffset</i>) from the first byte of the sort info block and the number of bytes (<i>dataSize</i>) to write. This function writes <i>dataSize</i> bytes to the sort info block from the buffer pointed to by <i>pSortInfoBlock</i>. The total sort info block size after a successful write is <i>dataOffset</i> + <i>dataSize</i>. Any existing data following the <i>dataOffset</i> + <i>dataSize</i> position is discarded.</p> <p>To truncate the sort info block at a given position, specify <i>dataSize</i> = 0 and <i>dataOffset</i> > 0 (the position from which to truncate). The <i>dataOffset</i> value you specify becomes the new sort info block size.</p> <p>To delete the sort info block, specify zero for the <i>dataSize</i>, <i>dataOffset</i>, and <i>pSortInfoBlock</i> parameters.</p> <p>To append data to the end of existing sort info block data, set the <i>dataOffset</i> parameter to kOffsetEndOfData. Using this special offset is equivalent to specifying an offset equal to the data size, but does not require a prior call to retrieve the size when it is not known in advance.</p>
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncDmReadAppInfoBlock()

4

Classic Sync Manager API

The classic Sync Manager API consists of only those APIs that work with [classic databases](#).

This chapter has the following sections:

Classic Sync Manager Classes	290
Classic Sync Manager Structures and Types	304
Classic Sync Manager Constants	310
Classic Sync Manager Functions	311

All Sync Manager functions are available in Sync20.dll. The API described in this chapter is declared in SyncMgr.h.

Classic Sync Manager API

Classic Sync Manager Classes

Classic Sync Manager Classes

This section describes the the following classes, which only classic Sync Manager functions take as parameters, as described in “[Classic Sync Manager Functions](#)” on page 311. Additional classes used by classic as well as extended Sync Manager functions are described in “[Common Sync Manager Classes](#)” on page 384. These classes define only data members and no methods.

Functions introduced in Sync Manager version 2.4 do not use these classes.

CCallModuleParams	Defines information that SyncCallRemoteModule() passes to and receives from an application/module on the handheld.
CDBCreateDB	Defines information about a database to be created with the SyncCreateDB() function.
CDBGenInfo	Defines information about the application info or sort info blocks in a database header.
CPositionInfo	Defines how the SyncReadPositionXMap() function retrieves a list of record IDs in their sorted order from a database on the handheld.
CRawPreferenceInfo	Defines an application preferences block that is smaller than 64 Kbytes and is accessed with SyncReadAppPreference() and SyncWriteAppPreference().
CRawRecordInfo	Defines the structure of a record in a <i>classic</i> record or resource database.
CRawRecordInfo	Defines information about a database to be created with the SyncCreateDB() function.

CCallModuleParams

Purpose Defines information that [SyncCallRemoteModule\(\)](#) passes to and receives from an application/module on the handheld.

Declared In SyncMgr.h

Prototype

```
class CCallModuleParams {
public:
    UInt32 m_dwCreatorID;
    UInt32 m_dwTypeID;
    UInt16 m_wActionCode;
    UInt32 m_dwParamSize;
    void *m_pParam;
    UInt32 m_dwResultBufSize;
    void *m_pResultBuf;
    UInt32 m_dwresultCode;
    UInt32 m_dwActResultSize;
    UInt32 m_dwReserved;
}
```

Data Members	
m_dwCreatorID	Specifies the creator ID of the target handheld application.
m_dwTypeID	Specifies the type ID of the target handheld application.
m_wActionCode	Specifies the application-specific action code to perform.
m_dwParamSize	Specifies the number of bytes in the m_pParam array.
m_pParam	Specifies a pointer to the parameter block.
m_dwResultBufSize	Specifies the total number of bytes in the m_pResultBuf array.
m_pResultBuf	Specifies a pointer to the result buffer.
m_dwresultCode	Receives the result code returned by the handheld application/module.
m_dwActResultSize	Receives the actual size of the result data.

Classic Sync Manager API

CCallModuleParams

`m_dwReserved`

Reserved for future use. The caller must set this member to NULL (0) before calling the Sync Manager function.

Comments

The value of `m_dwActResultSize` is greater than that of `m_dwResultBufSize`, if your buffer is not large enough to accommodate all the results data. In this case, the function copies only `dwResultBufSize` bytes of data into the buffer.

Compatibility

Sync Manager version: 2.1 or later.
Palm OS version: Palm OS 2.0 or later.

See Also

[SyncCallRemoteModule\(\)](#)

CDbCreateDB

Purpose Defines information about a database to be created with the [SyncCreateDB\(\)](#) function.

Declared In SyncMgr.h

Prototype

```
class CDbCreateDB {  
    public:  
        HSByte m_FileHandle;  
        UInt32 m_Creator;  
        eDbFlags m_Flags;  
        HSByte m_CardNo;  
        char m_Name[SYNC_DB_NAMELEN] ;  
        UInt32 m_Type;  
        UInt16 m_Version;  
        UInt32 m_dwReserved;  
}
```

Data Members **m_FileHandle**
Receives the handle to the database that was created.

m_Creator
Specifies the 4-byte creator ID for the database to be created. Note that the creator ID must be registered on the PalmSource web site before an application is released. PalmSource, Inc. reserves values consisting of all lowercase characters.

m_Flags
An [eDbFlags](#) value that specifies several characteristics about the database to be created (whether it is a classic record or resource database or a schema database, whether it is to be backed up, and so on).

m_CardNo
Specifies the number of the memory card on which the database is to be stored. The first memory card in the system is card number 0, and subsequent card numbers are incremented by 1. Always specify 0, except on certain HandSpring handhelds, which are the only handhelds to support more than one Palm OS® memory card.

m_Name
Specifies the name of the database to be created as a character array of size SYNC_DB_NAMELEN.

Classic Sync Manager API

CDbCreateDB

m_Type

Specifies a 4-byte database type. Palm OS associates special behavior with several identifier values, including 'DATA', 'data', 'appl', 'panl', and 'libr'.

m_Version

Specifies an application-specific version number for the new database.

m_dwReserved

Reserved for future use. You must set this value to NULL (0) before calling the [SyncCreateDB\(\)](#).

Compatibility Sync Manager version: 2.0 or later.

Palm OS version: All.

See Also [SyncCreateDB\(\)](#)

CDbGenInfo

Purpose Defines information about the [application info](#) or sort info blocks in a database header.

Declared In SyncMgr.h

Prototype

```
class CDbGenInfo {
public:
    char m_FileName[SYNC_DB_NAMELEN];
    UInt16 m_TotalBytes;
    UInt16 m_BytesRead;
    HSByte *m_pBytes;
    UInt32 m_dwReserved;
}
```

Data Members

m_FileName
Specifies the name of a handheld database as a null-terminated string. Note: this member is not used with the [SyncReadDBAppInfoBlock\(\)](#), [SyncReadDBSortInfoBlock\(\)](#), [SyncWriteDBAppInfoBlock\(\)](#), or [SyncWriteDBSortInfoBlock\(\)](#) functions.

m_TotalBytes
Specifies the total number of bytes in the **m_pBytes** array. When reading, the caller must first fill in this member with the size of the buffer pointed to by **m_pBytes**. When writing, the caller must first set both this member and **m_BytesRead** to the size of the block to write.

m_BytesRead
Defines the number of useful bytes in the **m_pBytes** array. See the comments below for version-specific behavior.

m_pBytes
Specifies a pointer to the buffer that the caller has allocated for the application or sort info block data.

m_dwReserved
Reserved for future use. The caller must set this member to NULL (0) before calling the Sync Manager function.

Classic Sync Manager API

CDbGenInfo

Comments	When <i>reading</i> an application or sort info block, the caller must fill in the value of <code>m_TotalBytes</code> with the size of the buffer pointed to by <code>m_pBytes</code> . The Sync Manager function fills in <code>m_BytesRead</code> with the actual size of the application or sort info block, which can be greater than the amount of data it copies to the <code>m_pBytes</code> buffer, if the application or sort info block is larger than the buffer. In this case, versions 2.1 and later of Sync Manager copy only the first <code>m_TotalBytes</code> of data to the caller's buffer; versions earlier than 2.1 copy <i>nothing</i> to the caller's buffer. When <i>writing</i> an application or sort info block, the caller must first set both <code>m_TotalBytes</code> and <code>m_BytesRead</code> to the size of the data to write.
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All. See the "Comments" section for version-based behavior differences.
See Also	SyncReadDBAppInfoBlock() , SyncReadDBSortInfoBlock() , SyncWriteDBAppInfoBlock() , SyncWriteDBSortInfoBlock()

CPositionInfo

Purpose Defines how the [SyncReadPositionXMap\(\)](#) function retrieves a list of record IDs in their sorted order from a database on the handheld.

Declared In SyncMgr.h

Prototype

```
class CPositionInfo {
    public:
        HSByte m_FileHandle;
        UInt16 m_FirstPos;
        UInt16 m_MaxEntries;
        UInt16 m_NumReadIn;
        UInt16 m_TotalBytes;
        HSByte *m_pBytes;
}
```

Data Members

m_FileHandle
Specifies the handle of an open classic database.

m_FirstPos
Specifies the index of the first record ID to read. This index is zero based.

m_MaxEntries
Specifies the total number of record IDs to read. Note that this value must be consistent with the value of the **m_TotalBytes** member: because each record ID is four bytes long, the value of **m_TotalBytes** must equal **n_MaxEntries * 4**.

m_NumReadIn
Receives the number of record IDs, not bytes, passed back in the **m_pBytes** array.

m_TotalBytes
Specifies the size, in bytes, of the buffer pointed to by **m_pBytes**. Note that this value must be consistent with the value of the **m_MaxEntries** member: because each record ID is four bytes long, the value of **m_TotalBytes** must equal **n_MaxEntries * 4**.

m_pBytes
Specifies a pointer to an array of bytes to receive record IDs—which the caller must allocate before calling [SyncReadPositionXMap\(\)](#).

Classic Sync Manager API

CPositionInfo

Compatibility Sync Manager version: 2.0 or later.

Palm OS version: All.

See “[Sync Manager API, Version 2.4 Changes](#)” on page 1110.

See Also [SyncReadPositionXMap\(\)](#)

CRawPreferenceInfo

Purpose Defines an [application preferences](#) block that is smaller than 64 Kbytes and is accessed with [SyncReadAppPreference\(\)](#) and [SyncWriteAppPreference\(\)](#).

Declared In SyncMgr.h

Prototype

```
class CRawPreferenceInfo {
    public:
        UInt16 m_version;
        UInt32 m_creator;
        UInt16 m_prefId;
        UInt16 m_reqBytes;
        UInt16 m_retBytes;
        UInt16 m_actSize;
        SInt32 m_backedUp;
        SInt32 m_nBytes;
        HSByte *m_pBytes;
        UInt32 m_dwReserved;
}
```

Data Members `m_version`

The preference version. When writing, this member is specified by the caller. When reading, it is received from the handheld.

`m_creator`

Specifies the preference creator ID to access. This is usually the same as the application's creator ID.

`m_prefId`

Specifies the preference ID to access.

`m_reqBytes`

When reading, this member specifies the maximum number of preference bytes requested (a value of 0xFFFF requests all the preference information). This value must not exceed the value of `m_nBytes`. When writing, this member is ignored.

`m_retBytes`

When reading, this member receives the number of preference bytes that were copied to the caller's buffer. When writing, this member is ignored.

Classic Sync Manager API

CRawPreferenceInfo

`m_actSize`

When reading, this member receives the actual size of the preference information on the handheld. This value will be larger than `m_retBytes` if the caller's buffer was not large enough. When writing, this member is ignored.

`m_backedUp`

Specifies whether the Sync Manager function accesses the *Saved* preferences database (TRUE), which is backed up during a HotSync operation, or the *Unsaved* preferences database (FALSE), which is not backed up.

`m_nBytes`

When reading, this member specifies the size, in bytes, of the `m_pBytes` array allocated by the caller. When writing, this member specifies the number of bytes of preference data to write.

`m_pBytes`

Specifies a pointer to an array of bytes to use as a data buffer for preference information. When reading, this buffer receives the requested data. When writing, this buffer holds the data to write. In either case, the caller must allocate this buffer before calling the Sync Manager function.

`m_dwReserved`

Reserved for future use. The caller must set this member to NULL (0) before calling the Sync Manager function.

Compatibility

Sync Manager version: 2.0 or later.
Palm OS version: 2.0 or later.

See Also

[SyncReadAppPreference\(\)](#), [SyncWriteAppPreference\(\)](#)

CRawRecordInfo

Purpose	Defines the structure of a record in a <i>classic</i> record or resource database.
Declared In	<code>SyncMgr.h</code>
Prototype	<pre>class CRawRecordInfo { public: HSByte m_FileHandle; UInt32 m_RecId; UInt16 m_RecIndex; HSByte m_Attrbs; SInt16 m_CatId; SInt32 m_ConduitId; UInt32 m_RecSize; UInt16 m_TotalBytes; HSByte *m_pBytes; UInt32 m_dwReserved; }</pre>
Data Members	<p>m_FileHandle Specifies the handle of an open classic database on the handheld. This is a handle returned by the SyncCreateDB() or SyncOpenDB() functions.</p> <p>m_RecId Defines the record ID for a classic record database, or the resource type for a resource database. When reading or deleting a record by ID, the caller specifies this member. When reading a record by index (or iteration), this member receives the record ID from the handheld. When deleting a resource, this member specifies the resource type. When writing, this member receives a record ID, which is always assigned by the handheld.</p> <p>m_RecIndex Defines the record index. When reading records or resources by index, the caller specifies this member. When reading a resource, this member receives the resource ID from the handheld. When deleting a resource, this member specifies the resource ID. When reading a record by ID using Sync Manager versions 2.1 and later, this member receives the record index from the handheld. For Sync Manager versions earlier than 2.1, this value is undefined.</p>

Classic Sync Manager API

CRawRecordInfo

`m_Attrbs`

Defines the attributes of the record. When reading, this member receives the attributes from the handheld. When writing, this member specifies the attributes to write. This member is a combination of one or more [`eSyncRecAttrs`](#) values.

`m_CatId`

Defines the index (0 to 15) of the category the record belongs to. When reading, this member receives the category index from the handheld. When writing, this member specifies the category to assign the record to.

`m_ConduitId`

This member has been deprecated.

`m_RecSize`

Defines the number of bytes of valid record or resource data in the `m_pBytes` array. When reading, this member receives the actual record or resource size. When writing, this member specifies the size of the data to write. See the comments below for details.

`m_TotalBytes`

Specifies the size of the `m_pBytes` buffer that the caller has allocated for record or resource data. When reading, this member specifies the size of the buffer. When writing, this member specifies the size of the record or resource to write. See the comments below for details.

`m_pBytes`

Specifies a pointer to an array of bytes to use as a data buffer for record or resource data. When reading, this buffer receives the requested data. When writing, this buffer holds the data to write. In either case, the caller must allocate this buffer before calling the Sync Manager function.

`m_dwReserved`

Reserved for future use. The caller must set this member to `NULL` (0) before calling the Sync Manager function.

Comments Many of the Sync Manager functions use objects of the CRawRecordInfo class to exchange *classic* database records and resources with the handheld. Each of the functions require that you fill in certain of the object members with information before calling the function; which members must be filled in are indicated in the documentation for each function.

When *reading* a record or resource, the caller must fill in the value of m_TotalBytes with the size of the buffer pointed to by m_pBytes. The Sync Manager function fills in m_RecSize with the actual size of the record or resource data on the handheld, which can be greater than the buffer size m_TotalBytes, if the record or resource is larger than the buffer. In this case, versions 2.1 and later of Sync Manager copy only the first m_TotalBytes of data to the caller's buffer; versions earlier than 2.1 copy *nothing* to the caller's buffer.

When *writing* record or resource data, the caller must first set both m_TotalBytes and m_RecSize to the size of the data to write.

NOTE: The buffer size m_TotalBytes is a UInt16 value, so only records and resources *smaller than 64 KB* can be read and written using CRawRecordInfo objects. The maximum record size in a classic database is 65,505 bytes.

Compatibility Sync Manager version: 2.0 or later.

Palm OS version: All.

See Also [SyncDeleteRec\(\)](#), [SyncDeleteResourceRec\(\)](#),
[SyncReadNextModifiedRec\(\)](#),
[SyncReadNextModifiedRecInCategory\(\)](#),
[SyncReadNextRecInCategory\(\)](#), [SyncReadRecordById\(\)](#),
[SyncReadRecordByIndex\(\)](#),
[SyncReadResRecordByIndex\(\)](#), [SyncWriteRec\(\)](#),
[SyncWriteResourceRec\(\)](#)

Classic Sync Manager Structures and Types

This section describes the following data structures and data types, which you use only with the functions described in “[Classic Sync Manager Functions](#)” on page 311. Additional structures and types used by classic as well as extended and schema Sync Manager functions are described in “[Common Sync Manager Structures and Types](#)” on page 400.

[SyncDatabaseInfoType](#)

Receives information about a database on the handheld.

[SyncFindDbByNameParams](#)

Specifies information used for finding a database with `SyncFindDbByName()`.

[SyncFindDbByTypeCreatorParams](#)

Specifies information used for finding a database with `SyncFindDbByTypeCreator()`.

[SyncReadOpenDbInfoParams](#)

Specifies information for retrieving handheld database information with `SyncReadOpenDbInfo()`.

SyncDatabaseInfoType Struct

Purpose	Receives information about a database on the handheld.
Declared In	<code>SyncMgr.h</code>
Prototype	<pre>typedef struct SyncDatabaseInfoType { CDbList baseInfo; UInt32 dwNumRecords; UInt32 dwTotalBytes; UInt32 dwDataBytes; UInt32 dwAppBlkSize; UInt32 dwSortBlkSize; UInt32 dwMaxRecSize; UInt32 dwReserved; UInt32 dwLocalID; UInt32 dwOpenRef; } SyncDatabaseInfoType</pre>
Fields	<p>baseInfo Receives a CDbList object containing basic information about a database.</p> <p>dwNumRecords Receives the number of records or resources in a database. This field is filled in only if the <code>SYNC_DB_INFO_OPT_GET_SIZE</code> flag is set.</p> <p>dwTotalBytes Receives the total bytes of storage used by a database, including overhead. This field is filled in only if the <code>SYNC_DB_INFO_OPT_GET_SIZE</code> flag is set.</p> <p>dwDataBytes Receives the total bytes of storage used by a database for data, excluding overhead. This field is filled in only if the <code>SYNC_DB_INFO_OPT_GET_SIZE</code> flag is set.</p> <p>dwAppBlkSize Receives the block size, in bytes, of the application info block. This field is filled in only for SyncReadOpenDbInfo() and only if the <code>SYNC_DB_INFO_OPT_GET_SIZE</code> option is set.</p> <p>dwSortBlkSize Receives the block size, in bytes, of the sort info block. This field is filled in only for SyncReadOpenDbInfo() and only if the <code>SYNC_DB_INFO_OPT_GET_SIZE</code> option is set.</p>

Classic Sync Manager API

SyncDatabaseInfoType

dwMaxRecSize

Receives the size of the largest record or resource in the database. This field is filled in only for [SyncReadOpenDbInfo\(\)](#) and only if the SYNC_DB_INFO_OPT_GET_MAX_REC_SIZE option is set.

dwReserved

Reserved for future use. The caller must set this field to NULL (0) before calling the Sync Manager function.

dwLocalID

Receives a value for Sync Manager's internal use only.

dwOpenRef

Receives a value for Sync Manager's internal use only.

Compatibility Sync Manager version: 2.2 or later.
Palm OS version: 3.0 or later.

See Also [SyncFindDbByName\(\)](#), [SyncFindDbByTypeCreator\(\)](#),
[SyncReadOpenDbInfo\(\)](#).

SyncFindDbByNameParams Struct

Purpose Specifies information used for finding a database with [SyncFindDbByName \(\)](#).

Declared In SyncMgr.h

Prototype

```
typedef struct SyncFindDbByNameParams {  
    HSBYTE bOptFlags;  
    UInt32 dwCardNum;  
    char *pcDatabaseName;  
} SyncFindDbByNameParams
```

Fields bOptFlags

Specifies the option flags for the find operation. The caller can combine values from the [Database Information Retrieval Options](#) (SYNC_DB_INFO_OPT_GET_...).

dwCardNum

Specifies the number of the memory card on which the database is stored. The first memory card in the system is card number 0, and subsequent card numbers are incremented by 1. Always specify 0, except on certain HandSpring handhelds, which are the only handhelds to support more than one Palm OS memory card.

pcDatabaseName

Specifies the database name as a pointer to a null-terminated string.

Compatibility Sync Manager version: 2.2 or later.

Palm OS version: 3.0 or later.

See Also [SyncFindDbByName \(\)](#)

Classic Sync Manager API

SyncFindDbByTypeCreatorParams

SyncFindDbByTypeCreatorParams Struct

Purpose	Specifies information used for finding a database with SyncFindDbByTypeCreator() .
Declared In	SyncMgr.h
Prototype	<pre>typedef struct SyncFindDbByTypeCreatorParams { HSByte bOptFlags; HSByte bSrchFlags; UInt32 dwType; UInt32 dwCreator; } SyncFindDbByTypeCreatorParams</pre>
Fields	<p>bOptFlags Specifies the option flags for the find operation. The caller can combine values from the Database Information Retrieval Options (SYNC_DB_INFO_OPT_GET_...).</p> <p>bSrchFlags Specifies the search options for finding the database. The caller can combine values from the Database Search Options (SYNC_DB_SRCH_OPT_...).</p> <p>dwType Specifies the database type as a 4-byte type identifier. The caller can specify a value of 0 to perform a wildcard search for any database type.</p> <p>dwCreator Specifies the creator ID of the database to find. Specify a value of 0 to search for any creator ID.</p>
Compatibility	Sync Manager version: 2.2 or later. Palm OS version: 3.0 or later.
See Also	SyncFindDbByTypeCreator()

SyncReadOpenDbInfoParams Struct

Purpose	Specifies information for retrieving handheld database information with SyncReadOpenDbInfo() .
Declared In	SyncMgr.h
Prototype	<pre>typedef struct SyncReadOpenDbInfoParams { HSByte bOptFlags; HSByte bDbHandle; } SyncReadOpenDbInfoParams</pre>
Fields	bOptFlags Specifies the option flags for the find operation. The caller can combine values from the Database Information Retrieval Options (SYNC_DB_INFO_OPT_GET_...). bDbHandle Specifies a handle to an open database, as returned from a call to SyncOpenDB() or SyncCreateDB() .
Compatibility	Sync Manager version: 2.2 or later. Palm OS version: 3.0 or later.
See Also	SyncFindDbByTypeCreator()

Classic Sync Manager Constants

This section describes the following preprocessor constants, which you use only with the functions described in “[Classic Sync Manager Functions](#)” on page 311. Additional constants used by classic as well as extended and schema Sync Manager functions are described in “[Common Sync Manager Constants](#)” on page 405.

Deprecated Constants	Deprecated versions of newer constants used in <code>CRawRecordInfo.m_Attribs</code> . Use the values defined by the eSyncRecAttrs enum instead.
--------------------------------------	--

Deprecated Constants

Purpose	Deprecated versions of newer constants used in <code>CRawRecordInfo.m_Attribs</code> . Use the values defined by the eSyncRecAttrs enum instead, as shown below.
Declared In	<code>SyncMgr.h</code>
Constants	<code>#define ARCHIVE_BIT eRecAttrArchived</code> <code>#define DELETE_BIT eRecAttrDeleted</code> <code>#define DIRTY_BIT eRecAttrDirty</code> <code>#define PRIVATE_BIT eRecAttrSecret</code>
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.
See Also	eSyncRecAttrs , CRawRecordInfo

Classic Sync Manager Functions

This section describes the following functions, which access only classic databases. See “[Common Sync Manager Functions and Macros](#)” on page 434 for functions that access all types of databases or utility functions and macros that do not access databases.

[SyncCallRemoteModule\(\)](#)

Calls a module (an application, panel, or other executable) on the handheld and returns data and status information to your conduit from that module.

[SyncChangeCategory\(\)](#)

Moves all records from one category (the source category) to another category (the target category) in a classic database.

[SyncCloseDB\(\)](#)

Closes a classic database that was opened by SyncOpenDB() or SyncCreateDB().

[SyncCloseDBEx\(\)](#)

Closes a classic database that was previously opened by a call to SyncOpenDB() or SyncCreateDB() and optionally updates its backup and modification dates.

[SyncCreateDB\(\)](#)

Creates a new classic database on the handheld, opens the database, and returns a handle to it.

[SyncDeleteAllResourceRec\(\)](#)

Removes all resources from a classic database and disposes of their data.

[SyncDeleteDB\(\)](#)

Deletes a classic database on the handheld.

[SyncDeleteRec\(\)](#)

Removes a record from a classic database and disposes of its data.

[SyncDeleteResourceRec\(\)](#)

Removes a resource from a classic database and disposes of its data.

[SyncFindDbByName\(\)](#)

Searches for a classic database by name and memory card number on the handheld and returns information about the database, if it is found.

Classic Sync Manager API

Classic Sync Manager Functions

<u>SyncFindDbByTypeCreator()</u>	Searches for a classic database by type and creator ID on the handheld and returns information about the database, if it is found.
<u>SyncGetDBRecordCount()</u>	Retrieves the total record or resource count for a classic database on the handheld.
<u>SyncMaxRemoteRecSize()</u>	Retrieves the maximum record or resource size supported by classic databases on the handheld.
<u>SyncOpenDB()</u>	Opens an existing classic database on the handheld.
<u>SyncPurgeAllRecs()</u>	Removes all the records from a classic database and disposes of their data, regardless of record status.
<u>SyncPurgeAllRecsInCategory()</u>	Removes all the records in the specified category from a classic database and disposes of their data, regardless of record status.
<u>SyncPurgeDeletedRecs()</u>	Removes all the records marked as deleted or archived from a classic database and disposes of their data.
<u>SyncReadAppPreference()</u>	Retrieves an application's preferences block from one of the application preferences databases on the handheld.
<u>SyncReadDBAppInfoBlock()</u>	Retrieves the application info block, if one exists, from a classic database on the handheld.
<u>SyncReadDBList()</u>	Retrieves information about a list of classic databases on the handheld.
<u>SyncReadDBSortInfoBlock()</u>	Retrieves the sort info block, if one exists, from a classic database on the handheld.

[SyncReadNextModifiedRec\(\)](#)

Retrieves the next modified, archived, or deleted record from a classic record database on the handheld. Each call to this iterator function retrieves the next modified record until all modified records have been returned.

[SyncReadNextModifiedRecInCategory\(\)](#)

Retrieves the next modified record, including deleted and archived records, in a category from a classic database on the handheld. Each call to this iterator function retrieves the next modified record in a category, until all modified records have been retrieved.

[SyncReadNextRecInCategory\(\)](#)

Retrieves any record, including deleted, archived, and modified records, in a category from a classic record database on the handheld. Each call to this iterator function retrieves the next record in a category, until all records have been retrieved.

[SyncReadOpenDbInfo\(\)](#)

Retrieves comprehensive information about a classic database on the handheld.

[SyncReadPositionXMap\(\)](#)

Retrieves a list of the record IDs in their sorted order from a classic database on the handheld.

[SyncReadRecordById\(\)](#)

Retrieves a record, by unique record ID, from a classic record database on the handheld.

[SyncReadRecordByIndex\(\)](#)

Retrieves a record, by index, from a classic record database on the handheld.

[SyncReadResRecordByIndex\(\)](#)

Retrieves a resource record, by index, from an open resource database on the handheld.

[SyncResetRecordIndex\(\)](#)

Resets the record iteration index of an open, classic record database on the handheld.

Classic Sync Manager API

Classic Sync Manager Functions

[SyncResetSyncFlags\(\)](#)

Resets the modified flag of all records in the open, classic record database on the handheld. Resets the backup date for an open, classic record or resource database.

[SyncWriteAppPreference\(\)](#)

Writes application preferences into a preferences database on the handheld.

[SyncWriteDBAppInfoBlock\(\)](#)

Writes an application info block to a classic record or resource database on the handheld.

[SyncWriteDBSortInfoBlock\(\)](#)

Writes a sort information block to an classic record or resource database on the handheld.

[SyncWriteRec\(\)](#)

Writes a record to a classic record database on the handheld.

[SyncWriteResourceRec\(\)](#)

Writes a resource to a classic resource database on the handheld.

SyncCallRemoteModule Function

Purpose	Calls a module (an application, panel, or other executable) on the handheld and returns data and status information to your conduit from that module.
Declared In	<code>SyncMgr.h</code>
Prototype	<code>SInt32 SyncCallRemoteModule (CCcallModuleParams *pParams)</code>
Parameters	$\leftrightarrow pParams$ An object of the CCallModuleParams class, which specifies information for the called module and receives information from the called module. Note that there is a 64-KB limit on the size of the argument passed from the desktop to the handheld application.
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: <code>SYNCERR_COMM_NOT_INIT</code> <code>SYNCERR_LOST_CONNECTION</code> <code>SYNCERR_REMOTE_SYS</code> <code>SYNCERR_REMOTE_MEM</code> <code>SYNCERR_UNKNOWN_REQUEST</code> The module on the handheld was not found, or the module on did not handle the action code. <code>SYNCERR_LOCAL_BUFF_TOO_SMALL</code> The results buffer was not large enough to contain the results data. If this is the case, then upon return the value of <code>m_dwActResultSize</code> is greater than the value of <code>m_dwResultBufSize</code> , and only <code>m_dwResultBufSize</code> bytes were copied to the results buffer. See “ Common Sync Manager Error Codes ” on page 486 for a description of all Sync Manager error codes.

Classic Sync Manager API

SyncCallRemoteModule

Comments

IMPORTANT: This function works only with classic databases, which is how [68K applications](#) are stored. It works on both Palm OS Cobalt and Palm OS Garnet or earlier handhelds, but on Palm OS Cobalt handhelds it can call only a classic database. To call a [Palm OS Protein application](#) on a Palm OS Cobalt handheld, you must use [SyncCallDeviceApplication\(\)](#) instead.

Most conduits can accomplish their jobs without using this function. PalmSource recommends not using this function unless absolutely essential.

This function enables the caller to send arbitrary data in the *pParams* parameter to a module on the handheld during a HotSync operation. The module can pass back variable-sized information in *pParams*, which the caller can examine when this function returns.

Note that the format of the data and the action codes are completely module-specific. The handheld module that you call must have the same structure as a Palm OS application; however, the module can have a proprietary type ID so that it does not show up in the Launcher.

When you call `SyncCallRemoteModule()` from a conduit, the module on the handheld launches with a `sysAppLaunchCmdHandleSyncCallApp` launch code. To handle this launch code, the handheld module must cast the command parameter block passed to `PilotMain()` to a `SysAppLaunchCmdHandleSyncCallAppType` pointer. The `SysAppLaunchCmdHandleSyncCallAppType` structure contains all the information that the caller passed into `SyncCallRemoteModule()` on the desktop as well as the necessary fields to pass the result back to the desktop.

After the handheld module processes the `sysAppLaunchCmdHandleSyncCallApp` launch code, it must send a `DlkCallAppReplyParamType` reply back to the desktop using the DLServer's (`DLServer.h`) `DlkControl()` function.

[Table 4.1](#) and [Table 4.2](#) are some important mappings from the `CCallModuleParams` class of Sync Manager on the desktop to the `SysAppLaunchCmdHandleSyncCallAppType` and `DlkCallAppReplyParamType` structures on the handheld.

**Table 4.1 Mapping from desktop Sync Manager
CCallModuleParams to handheld
SysAppLaunchCmdHandleSyncCallAppType**

CCallModuleParams	SysAppLaunchCmdHandleSyncCallAppType structure
m_wActionCode	action
m_dwParamSize	dwParamSize
m_pParam	paramP

**Table 4.2 Mapping from desktop Sync Manager
CCallModuleParams to handheld
DlkCallAppReplyParamType**

CCallModuleParams	DlkCallAppReplyParamType structure
m_dwResultBufSize	dwResultSize
m_pResultBuf	resultP
m_dwresultCode	dwresultCode

For more information and example handheld application code, see *Exploring Palm OS: System Management*.

For Palm OS Cobalt Handhelds

When *pParams->m_dwTypeID* is zero, [*SyncCallRemoteModule\(\)*](#) launches the first application returned by *DmFindDatabaseByTypeCreator* (*dmFindClassicDB*) with type *sysFileTApplication*; otherwise if there is no such application, it launches the first application returned by *DmFindDatabaseByTypeCreator* (*dmFindClassicDB*) with type *sysFileTPanel*; otherwise it returns an error. So *SyncCallRemoteModule()* actually treats setting *m_dwTypeID* to zero as a substitute for first calling with a type ID of *sysFileTApplication* then a type ID of *sysFileTPanel*.

Classic Sync Manager API

SyncCallRemoteModule

As noted above, this function calls only classic databases; it cannot be used to call a [Palm OS Protein application](#) on a Palm OS Cobalt handheld.

Compatibility	Sync Manager version: 2.1 or later. Palm OS version: Palm OS 2.0 or later. See “Comments.”
	On a handheld running Palm OS Cobalt, version 6.0 and a desktop running Sync Manager 2.4 or earlier, if both an ARM-native application (extended resource database) and a 68K application (classic resource database) with the same name and creator ID are present on the handheld, you cannot specify which application that this function calls. Therefore the application that this function calls is indeterminate in this situation. This problem has been fixed in Sync Manager 2.5 so that this function calls only classic databases.

See Also	CCallModuleParams , SyncCallDeviceApplication()
-----------------	---

SyncChangeCategory Function

Purpose Moves all records from one category (the source category) to another category (the target category) in a classic database.

Declared In SyncMgr.h

Prototype SInt32 SyncChangeCategory (HSByte *fHandle*,
HSByte *from*, HSByte *to*)

Parameters → *fHandle*
A handle to a classic database on the handheld. This handle is returned by a call to [SyncOpenDB\(\)](#) or [SyncCreateDB\(\)](#). The database must be opened for reading and writing.

→ *from*
The index of the source category. This must be a value between 0 and 15.

→ *to*
The index of the target category. This must be a value between 0 and 15.

Returns If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following error code values:

SYNCERR_COMM_NOT_INIT

SYNCERR_LOST_CONNECTION

SYNCERR_REMOTE_BAD_ARG

SYNCERR_NO_FILES_OPEN

SYNCERR_BAD_OPERATION

SYNCERR_READ_ONLY

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Classic Sync Manager API

SyncChangeCategory

- Comments** This function changes the category index of all records in a specified category in an open database on the handheld, but it does not alter the modified status of the records.
- The category index values specified by *from* and *to* must be in the range 0 to 15. The convention is that index 0 is the unfiled category and index values 1 through 15 are filed category index values.
- Compatibility** Sync Manager version: 2.0 or later.
Palm OS version: All.

SyncCloseDB Function

Purpose	Closes a classic database that was opened by SyncOpenDB() or SyncCreateDB() .
Declared In	SyncMgr.h
Prototype	SInt32 SyncCloseDB (HSByte <i>fHandle</i>)
Parameters	→ <i>fHandle</i> A handle to a classic database on the handheld. This handle is returned by the call to SyncOpenDB() or SyncCreateDB() that opened this database.
Returns	If successful, returns SYNCERR_NONE, which means that the database was closed and its handle was destroyed. If unsuccessful, returns one of the following error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_MEM SYNCERR_REMOTE_BAD_ARG SYNCERR_NO_FILES_OPEN See “ Common Sync Manager Error Codes ” on page 486 for a description of all Sync Manager error codes.
Comments	The Sync Manager allows only one <i>classic</i> database to be open at any time; therefore you must use this function or SyncCloseDBEx() before opening another classic database or exiting your conduit. Otherwise, other conduits will not be able to open their classic databases.
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.
See Also	SyncCreateDB() , SyncOpenDB() , SyncCloseDBEx()

Classic Sync Manager API

SyncCloseDBEx

SyncCloseDBEx Function

Purpose	Closes a classic database that was previously opened by a call to SyncOpenDB() or SyncCreateDB() and optionally updates its backup and modification dates.
Declared In	<code>SyncMgr.h</code>
Prototype	<code>SInt32 SyncCloseDBEx (HSByte dbHandle, HSByte bOptFlags)</code>
Parameters	<p>→ <i>dbHandle</i> A handle to a classic database on the handheld. This handle is returned by the call to SyncOpenDB() or SyncCreateDB() that opened this database.</p> <p>→ <i>bOptFlags</i> Options for updating the database dates. You can combine the constants defined in “Database Closing Options” on page 410.</p>
Returns	<p>If successful, returns <code>SYNCERR_NONE</code>, which means that the database was closed and its handle was destroyed.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p style="margin-left: 40px;"><code>SYNCERR_COMM_NOT_INIT</code> <code>SYNCERR_LOST_CONNECTION</code> <code>SYNCERR_REMOTE_SYS</code> <code>SYNCERR_REMOTE_MEM</code> <code>SYNCERR_REMOTE_BAD_ARG</code> <code>SYNCERR_NO_FILES_OPEN</code></p> <p>See “Common Sync Manager Error Codes” on page 486 for a description of all Sync Manager error codes.</p>

Comments	This function is an extended version of the SyncCloseDB() function. <code>SyncCloseDBEx()</code> takes an additional option flags parameter (<i>bOptFlags</i>) and can update the modification and backup dates of the database while closing it. If the handheld is using a version of Palm OS earlier than version 3.0, you must specify a value of 0 for the <i>bOptFlags</i> argument; otherwise, this function fails. The Sync Manager allows only one <i>classic</i> database to be open at any time; therefore you must use this function or SyncCloseDB() before opening another classic database or exiting your conduit. Otherwise, other conduits will not be able to open their classic databases.
Compatibility	Sync Manager version: 2.2 or later. Palm OS version: All*.
See Also	SyncCreateDB() , SyncOpenDB() , SyncCloseDB()

Classic Sync Manager API

SyncCreateDB

SyncCreateDB Function

Purpose	Creates a new classic database on the handheld, opens the database, and returns a handle to it.
Declared In	SyncMgr.h
Prototype	SInt32 SyncCreateDB (CDBCreateDB &rDbStats)
Parameters	$\leftrightarrow rDbStats$ Database creation information specified in an object of the CDBCreateDB class. The <code>m_FileHandle</code> data member of this parameter receives a handle to the open database.
Returns	If successful, returns SYNCERR_NONE, which means that the database was created and its handle stored into the <code>m_FileHandle</code> data member of <code>rDbStats</code> . If unsuccessful, returns one of the following error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_MEM SYNCERR_REMOTE_BAD_ARG SYNCERR_FILE_ALREADY_EXIST SYNCERR_TOO_MANY_OPEN_FILES SYNCERR_FILE_NOT_OPEN See “ Common Sync Manager Error Codes ” on page 486 for a description of all Sync Manager error codes.

Comments	<p>After this function creates the database, it opens the database for exclusive read-write access, with private (secret) records shown.</p> <p><code>SyncCreateDB()</code> cannot overwrite an existing database. If you attempt to create a database with the name of an existing database, <code>SyncCreateDB</code> fails with the <code>SYNCERR_FILE_ALREADY_EXIST</code> error. To replace an existing database, explicitly delete the old database with <code>SyncDeleteDB()</code> first and then call <code>SyncCreateDB()</code> to create the new database.</p> <p>The Sync Manager allows only one <i>classic</i> database to be open at any time; thus, you must close any opened classic database before calling this function. If you open a classic database, you must close it before exiting your conduit; otherwise, other conduits will not be able to open their classic databases.</p>
Compatibility	<p>Sync Manager version: 2.0 or later. Palm OS version: All.</p>
See Also	<p><code>CDbCreateDB</code>, <code>SyncDeleteDB()</code>, <code>SyncCloseDB()</code>, <code>SyncCloseDBEx()</code></p>

Classic Sync Manager API

SyncDeleteAllResourceRec

SyncDeleteAllResourceRec Function

Purpose	Removes all resources from a classic database and disposes of their data.
Declared In	SyncMgr.h
Prototype	SInt32 SyncDeleteAllResourceRec (HSByte <i>fHandle</i>)
Parameters	<i>fHandle</i> A handle to the classic database on the handheld. This handle is returned by a call to SyncOpenDB() or SyncCreateDB() .
Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_BAD_ARG SYNCERR_NO_FILES_OPEN SYNCERR_BAD_OPERATION SYNCERR_READ_ONLY See “ Common Sync Manager Error Codes ” on page 486 for a description of all Sync Manager error codes.
Comments	Before calling this function, the database must be open for reading and writing. This function immediately disposes of the resources’ data chunks and removes the resources’ entries from the database header.
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.
See Also	SyncOpenDB() , SyncCreateDB()

SyncDeleteDB Function

Purpose	Deletes a classic database on the handheld.
Declared In	<code>SyncMgr.h</code>
Prototype	<code>SInt32 SyncDeleteDB (const char *pName, SInt32 nCardNum)</code>
Parameters	<p>→ <i>pName</i> The name of the database on the handheld. This is a null-terminated array of characters.</p> <p>→ <i>nCardNum</i> The number of the memory card on which the database resides on the handheld. The first memory card in the system is card number 0, and subsequent card numbers are incremented by 1. Always specify 0, except on certain HandSpring handhelds, which are the only handhelds to support more than one Palm OS memory card.</p>
Returns	<p>If successful, returns <code>SYNCERR_NONE</code>, which means that the database was deleted.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p><code>SYNCERR_COMM_NOT_INIT</code> <code>SYNCERR_LOST_CONNECTION</code> <code>SYNCERR_REMOTE_SYS</code> <code>SYNCERR_REMOTE_MEM</code> <code>SYNCERR_REMOTE_BAD_ARG</code> <code>SYNCERR_NOT_FOUND</code> <code>SYNCERR_FILE_NOT_OPEN</code> <code>SYNCERR_FILE_ALREADY_OPEN</code></p> <p>See “Common Sync Manager Error Codes” on page 486 for a description of all Sync Manager error codes.</p>
Comments	You cannot delete an open database; you must close the database first.
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.
See Also	SyncCloseDB() , SyncCloseDBEx() , SyncCreateDB()

Classic Sync Manager API

SyncDeleteRec

SyncDeleteRec Function

Purpose	Removes a record from a classic database and disposes of its data.
Declared In	SyncMgr.h
Prototype	SInt32 SyncDeleteRec (CRawRecordInfo &rInfo)
Parameters	$\rightarrow rInfo$ An object of the CRawRecordInfo class, which specifies information about the record and database. The caller must specify the m_FileHandle and m_RecId members.
Returns	If successful, returns SYNCERR_NONE, which means that the record was destroyed. If unsuccessful, returns one of the following error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_MEM SYNCERR_REMOTE_BAD_ARG SYNCERR_NO_FILES_OPEN SYNCERR_BAD_OPERATION SYNCERR_READ_ONLY SYNCERR_NOT_FOUND See “ Common Sync Manager Error Codes ” on page 486 for a description of all Sync Manager error codes.

Comments	This function immediately disposes of the record's data chunk and removes the record's entry from the database header. The database must be open for reading and writing before a record can be deleted. To specify the record that you want deleted, create an object of the CRawRecordInfo class and fill in the object's <code>m_FileHandle</code> and <code>m_RecId</code> members. The record data and its entry in the database's record list are completely deleted. Note that the HotSync iteration index is not updated when you delete a record on a handheld that is running a version of Palm OS earlier than version 2.0. For a description of the possible difficulties with modifying a database while iterating through it, see " Modifying a Database While Iterating " on page 39 in the <i>C/C++ Sync Suite Companion</i> .
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All. See the "Comments" section for version-based behavior differences.
See Also	CRawRecordInfo

Classic Sync Manager API

SyncDeleteResourceRec

SyncDeleteResourceRec Function

Purpose	Removes a resource from a classic database and disposes of its data.
Declared In	SyncMgr.h
Prototype	SInt32 SyncDeleteResourceRec (CRawRecordInfo &rInfo)
Parameters	→ <i>rInfo</i> An object of the CRawRecordInfo class, which specifies information about the resource and database. The caller must specify <i>m_FileHandle</i> , <i>m_RecId</i> , and <i>m_RecIndex</i> members.
Returns	If successful, returns SYNCERR_NONE, which means that the resource was destroyed. If unsuccessful, returns one of the following error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_MEM SYNCERR_REMOTE_BAD_ARG SYNCERR_NO_FILES_OPEN SYNCERR_BAD_OPERATION SYNCERR_READ_ONLY SYNCERR_NOT_FOUND See “ Common Sync Manager Error Codes ” on page 486 for a description of all Sync Manager error codes.

Comments	This function immediately disposes of the resource's data chunk and removes the resource's entry from the database header. The database must be open for reading and writing before a resource can be deleted. To specify the resource that you want deleted, create an object of the CRawRecordInfo class and fill in the object's <code>m_FileHandle</code> , <code>m_RecId</code> , and <code>m_RecIndex</code> members.
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.
See Also	CRawRecordInfo

Classic Sync Manager API

SyncFindDbByName

SyncFindDbByName Function

Purpose Searches for a classic database by name and memory card number on the handheld and returns information about the database, if it is found.

Declared In SyncMgr.h

Prototype SInt32 SyncFindDbByName
(SyncFindDbByNameParams &rParam,
SyncDatabaseInfoType &rInfo)

Parameters → *rParam*
A structure of type [SyncFindDbByNameParams](#) that specifies the name, card number, and options for finding the database.

← *rInfo*
A structure of type [SyncDatabaseInfoType](#) that receives information about the found database.

Returns If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following error code values:

SYNCERR_COMM_NOT_INIT
SYNCERR_LOST_CONNECTION
SYNCERR_REMOTE_SYS
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_BAD_ARG
SYNCERR_NOT_FOUND

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments	To find a database by name, the caller fills in the structure pointed to by <i>rParam</i> with the database name and retrieval options, and this function passes back a structure with information about the database filled in. This function searches for a classic database only; it does not include extended or schema databases in its search.
Compatibility	Sync Manager version: 2.2 or later. Palm OS version: 3.0 or later.
See Also	SyncFindDbByNameParams , SyncDatabaseInfoType , SyncFindDbByTypeCreator()

Classic Sync Manager API

SyncFindDbByTypeCreator

SyncFindDbByTypeCreator Function

Purpose	Searches for a classic database by type and creator ID on the handheld and returns information about the database, if it is found.
Declared In	SyncMgr.h
Prototype	<pre>SInt32 SyncFindDbByTypeCreator (SyncFindDbByTypeCreatorParams &rParam, SyncDatabaseInfoType &rInfo)</pre>
Parameters	<p>→ <i>rParam</i> A structure of type SyncFindDbByTypeCreatorParams that specifies the database creator, type, and options for finding the database.</p> <p>← <i>rInfo</i> A structure of type SyncDatabaseInfoType that receives information about the found database.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_MEM SYNCERR_REMOTE_BAD_ARG SYNCERR_NOT_FOUND</p> <p>See “Common Sync Manager Error Codes” on page 486 for a description of all Sync Manager error codes.</p>

Comments	To find a database by type and creator ID, the caller fills in the structure pointed to by <i>rParam</i> with the database type and creator ID values and the retrieval options, and this function passes back a structure with information about the database filled in. You can use this function to enumerate through multiple databases of a particular type or creator ID. To begin a new search for a specific creator ID/type pair, the caller must specify the SYNC_DB_SRCH_OPT_NEW_SEARCH flag in the bSrchFlags field of <i>rParam</i> . Subsequent calls in the same sequence must exclude this flag. Note that this function does not support creation or deletion of databases in the middle of enumerating through them. This function searches for a classic database only; it does not include extended or schema databases in its search.
Compatibility	Sync Manager version: 2.2 or later. Palm OS version: 3.0 or later.
See Also	SyncFindDbByTypeCreatorParams , SyncDatabaseInfoType , SyncFindDbByName ()

Classic Sync Manager API

SyncGetDBRecordCount

SyncGetDBRecordCount Function

Purpose	Retrieves the total record or resource count for a classic database on the handheld.
Declared In	SyncMgr.h
Prototype	SInt32 SyncGetDBRecordCount (HSByte <i>fHandle</i> , UInt16 & <i>rNumRecs</i>)
Parameters	<p>→ <i>fHandle</i> A handle to a classic database on the handheld. This handle is returned by a call to SyncOpenDB() or SyncCreateDB().</p> <p>← <i>rNumRecs</i> The number of records or resources in the database.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_BAD_ARG SYNCERR_NO_FILES_OPEN</p> <p>See “Common Sync Manager Error Codes” on page 486 for a description of all Sync Manager error codes.</p>
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.
See Also	SyncOpenDB() , SyncCreateDB()

SyncMaxRemoteRecSize Function

Purpose	Retrieves the maximum record or resource size supported by classic databases on the handheld.
Declared In	<code>SyncMgr.h</code>
Prototype	<code>SInt32 SyncMaxRemoteRecSize (UInt32 &rdwMaxRecSize)</code>
Parameters	$\leftarrow rdwMaxRecSize$ The maximum record size, in bytes, that can be allocated on the handheld. This value is the maximum allowed size; the actual maximum size that can be allocated at a specific time is subject to available memory. Note the following special values: 0 Indicates that the maximum record size is unknown. 0xFFFFFFFF Indicates that there is no size limit, subject to available memory.
Returns	If successful, returns <code>SYNCERR_NONE</code> . If unsuccessful, returns one of the following error code values: <code>SYNCERR_COMM_NOT_INIT</code> <code>SYNCERR_LOST_CONNECTION</code> See “ Common Sync Manager Error Codes ” on page 486 for a description of all Sync Manager error codes.

Classic Sync Manager API

SyncMaxRemoteRecSize

Comments [Table 4.3](#) provides the maximum record size supported by versions of Palm OS.

Table 4.3 Maximum record size in classic databases

Palm OS Version	Maximum Record Size (bytes)
< 3.0	64,720
>= 3.0	65,505

NOTE: Sync Manager versions 2.4 and later return only the value 65,505 regardless of the version of Palm OS on the handheld.

Compatibility Sync Manager version: 2.0 or later.
Palm OS version: All.

SyncOpenDB Function

Purpose Opens an existing classic database on the handheld.

Declared In SyncMgr.h

Prototype

```
SInt32 SyncOpenDB (const char *pName,  
                  SInt32 nCardNum, HSByte &rHandle,  
                  HSByte openMode =  
                  (eDbWrite | eDbRead | eDbExclusive))
```

Parameters → *pName*

The name of the classic database to open. This is a null-terminated character array.

→ *nCardNum*

The number of the memory card on which the database resides. The first memory card in the system is card number 0, and subsequent card numbers are incremented by 1. Always specify 0, except on certain HandSpring handhelds, which are the only handhelds to support more than one Palm OS memory card.

← *rHandle*

The returned handle to the classic database.

→ *openMode*

A combination of one or more [eDbOpenModes](#) values to specify how to open the database—for example, open for reading and writing.

Returns If successful, returns SYNCERR_NONE, which means that the database was opened.

If unsuccessful, returns one of the following error code values:

SYNCERR_COMM_NOT_INIT

SYNCERR_LOST_CONNECTION

SYNCERR_REMOTE_SYS

SYNCERR_REMOTE_MEM

SYNCERR_REMOTE_BAD_ARG

SYNCERR_NOT_FOUND

SYNCERR_TOO_MANY_OPEN_FILES

SYNCERR_FILE_NOT_OPEN

Classic Sync Manager API

SyncOpenDB

SYNCERR_FILE_ALREADY_OPEN

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

- Comments** Sync Manager allows only one database to be open at any time; therefore you must close any open database before calling this function. If you open a database, you must close it before exiting your conduit; otherwise, other conduits will not be able to open their databases.
- Compatibility** Sync Manager version: 2.0 or later.
Palm OS version: All.
- See Also** [eDbOpenModes](#), [SyncCloseDB\(\)](#), [SyncCloseDBEx\(\)](#)

SyncPurgeAllRecs Function

Purpose Removes all the records from a classic database and disposes of their data, regardless of record status.

Declared In SyncMgr.h

Prototype SInt32 SyncPurgeAllRecs (HSByte *fHandle*)

Parameters → *fHandle*

A handle to a classic record database on the handheld. This handle is returned by a call to [SyncOpenDB\(\)](#) or [SyncCreateDB\(\)](#).

Returns If successful, returns SYNCERR_NONE. Also returns SYNCERR_NONE if the database has no records.

If unsuccessful, returns one of the following error code values:

SYNCERR_COMM_NOT_INIT

SYNCERR_LOST_CONNECTION

SYNCERR_REMOTE_SYS

SYNCERR_REMOTE_MEM

SYNCERR_REMOTE_BAD_ARG

SYNCERR_READ_ONLY

SYNCERR_BAD_OPERATION

SYNCERR_NO_FILES_OPEN

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments This function immediately disposes of all the records’ data chunks and removes the records’ entries from the database header.

The *fHandle* parameter must specify a classic database that is open for reading and writing.

Compatibility Sync Manager version: 2.0 or later.
Palm OS version: All.

See Also [SyncOpenDB\(\)](#), [SyncCreateDB\(\)](#),
[SyncPurgeAllRecsInCategory\(\)](#),
[SyncPurgeDeletedRecs\(\)](#)

Classic Sync Manager API

SyncPurgeAllRecsInCategory

SyncPurgeAllRecsInCategory Function

Purpose	Removes all the records in the specified category from a classic database and disposes of their data, regardless of record status.
Declared In	SyncMgr.h
Prototype	SInt32 SyncPurgeAllRecsInCategory (HSByte <i>fHandle</i> , SInt16 <i>category</i>)
Parameters	<p>→ <i>fHandle</i> A handle to a classic record database on the handheld. This handle is returned by a call to SyncOpenDB() or SyncCreateDB().</p> <p>→ <i>category</i> The index of the category whose records you want destroyed. By convention, use a value of 0 for the unfiled category, or use a value between 1 and 15 to specify a filed category.</p>
Returns	If successful, returns SYNCERR_NONE. Also returns SYNCERR_NONE if the database has no records. If unsuccessful, returns one of the following error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_MEM SYNCERR_REMOTE_BAD_ARG SYNCERR_NO_FILES_OPEN SYNCERR_BAD_OPERATION SYNCERR_READ_ONLY See “ Common Sync Manager Error Codes ” on page 486 for a description of all Sync Manager error codes.

Comments	For the specified database and category, this function immediately disposes of all the records' data chunks and removes the records' entries from the database header. The <i>fHandle</i> parameter must specify a classic database that is open for reading and writing. Note that this function does not update the record iteration index.
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: 2.0 or later.
See Also	SyncOpenDB() , SyncCreateDB()

Classic Sync Manager API

SyncPurgeDeletedRecs

SyncPurgeDeletedRecs Function

Purpose	Removes all the records marked as deleted or archived from a classic database and disposes of their data.
Declared In	SyncMgr.h
Prototype	SInt32 SyncPurgeDeletedRecs (HSByte <i>fHandle</i>)
Parameters	<p>→ <i>fHandle</i> A handle to a classic record database on the handheld. This handle is returned by a call to SyncOpenDB() or SyncCreateDB().</p>
Returns	<p>If successful, returns SYNCERR_NONE. Also returns SYNCERR_NONE if the database has no records.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_MEM SYNCERR_REMOTE_BAD_ARG SYNCERR_READ_ONLY SYNCERR_BAD_OPERATION SYNCERR_NO_FILES_OPEN</p> <p>See “Common Sync Manager Error Codes” on page 486 for a description of all Sync Manager error codes.</p>
Comments	<p>For all records whose delete bit is set, this function immediately disposes of all the records’ data chunks and removes the records’ entries from the database header.</p> <p>The <i>fHandle</i> parameter must specify a classic database that is open for reading and writing.</p>
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.
See Also	SyncOpenDB() , SyncCreateDB()

SyncReadAppPreference Function

Purpose	Retrieves an application's preferences block from one of the application preferences databases on the handheld.
Declared In	<code>SyncMgr.h</code>
Prototype	<code>SInt32 SyncReadAppPreference (CRawPreferenceInfo &rInfo)</code>
Parameters	<code>↔ rInfo</code> An object of the CRawPreferenceInfo class, which specifies and receives information about the application preferences.
Returns	If successful, returns <code>SYNCERR_NONE</code> . If unsuccessful, returns one of the following error code values: <code>SYNCERR_COMM_NOT_INIT</code> <code>SYNCERR_LOST_CONNECTION</code> <code>SYNCERR_REMOTE_SYS</code> <code>SYNCERR_REMOTE_MEM</code> <code>SYNCERR_REMOTE_BAD_ARG</code> <code>SYNCERR_UNKNOWN_REQUEST</code> <code>SYNCERR_NOT_FOUND</code> The requested preference was not found. See " Common Sync Manager Error Codes " on page 486 for a description of all Sync Manager error codes.
Comments	To use this function, allocate the <code>m_pBytes</code> buffer in an object of the CRawPreferenceInfo class and fill in the <code>m_creator</code> , <code>m_prefId</code> , <code>m_reqBytes</code> , <code>m_backedUp</code> , and <code>m_nBytes</code> members of the object. <code>SyncReadAppPreference()</code> fills in the <code>m_version</code> , <code>m_retBytes</code> , and <code>m_actSize</code> fields of the object and passes back the preferences data in the <code>m_pBytes</code> buffer.
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: 2.0 or later. Application preferences can be no larger than 64 KB, in Palm OS Cobalt as well as in Palm OS Garnet and earlier releases. Therefore

Classic Sync Manager API

SyncReadAppPreference

you can use this function to read preferences from handhelds running any of the above versions of Palm OS.

See Also [CRawPreferenceInfo](#), [SyncWriteAppPreference\(\)](#)

SyncReadDBAppInfoBlock Function

Purpose Retrieves the [application info](#) block, if one exists, from a classic database on the handheld.

Declared In SyncMgr.h

Prototype SInt32 SyncReadDBAppInfoBlock (HSByte *fHandle*,
CDBGenInfo &*rInfo*)

Parameters → *fHandle*
A handle to a classic record or resource database on the handheld. This handle is returned by a call to [SyncOpenDB\(\)](#) or [SyncCreateDB\(\)](#).

↔ *rInfo*
An object of the [CDBGenInfo](#) class, which specifies and receives information about the application info block and database.

Returns If successful, returns SYNCERR_NONE. Note that this function returns SYNCERR_NONE even if the buffer you allocated was too small for the resource, with version-specific details as follows:

- if you are using version 2.1 or later of the Sync Manager API, the first *m_TotalBytes* of block data is copied to your buffer, and the *m_BytesRead* field of *rInfo* is set to the actual block size.
- if you are using a version of the Sync Manager API earlier than version 2.1, nothing is copied to your buffer, but the *m_BytesRead* field of *rInfo* is set to the actual block size.

Because the Sync Manager does not generate an error for this condition, you must test for it upon function return with an error code of 0: if *m_BytesRead* is greater than *m_TotalBytes*, the buffer was too small.

If unsuccessful, returns one of the following error code values:

SYNCERR_COMM_NOT_INIT
SYNCERR_LOST_CONNECTION
SYNCERR_REMOTE_SYS
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_BAD_ARG

Classic Sync Manager API

SyncReadDBAppInfoBlock

SYNCERR_NOT_FOUND

The requested application info block is not available.

SYNCERR_NO_FILES_OPEN

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments

This function reads the application info block from a classic record or resource database that is open for reading. For more information about application info blocks, see “[Categories and the Application Info Block](#)” on page 145 in the *Introduction to Conduit Development*.

To use this function, allocate the `m_pBytes` buffer in an object of the `CDbGenInfo` class and fill in the `m_TotalBytes` member of the object. The `SyncReadDBAppInfo()` function fills in the remaining members of the object.

For performance optimization information, see “[Sync Manager and Performance](#)” on page 31 in the *C/C++ Sync Suite Companion*.

Compatibility

Sync Manager version: 2.0 or later.

Palm OS version: All.

See the “Result” section for version-based behavior differences.

See Also

[CDbGenInfo](#), [SyncOpenDB\(\)](#), [SyncCreateDB\(\)](#),
[SyncReadDBSortInfoBlock\(\)](#),
[SyncWriteDBAppInfoBlock\(\)](#)

SyncReadDBList Function

Purpose Retrieves information about a list of classic databases on the handheld.

Declared In SyncMgr.h

Prototype

```
SInt32 SyncReadDBList (HSByte cardNo,
                      UInt16 startIx, SInt32 bRam, CDbList *pList,
                      SInt32 &rCnt)
```

Parameters

→ *cardNo*
The memory card number on which the database(s) reside. The first memory card in the system is card number 0, and subsequent card numbers are incremented by 1. Always specify 0, except on certain HandSpring handhelds, which are the only handhelds to support more than one Palm OS memory card.

→ *startIx*
The starting index for the list of databases for which you want to retrieve information. Specify a value of 0 for the first database, and increment by 1 for each successive database.

→ *bRam*
A Boolean value. If this is TRUE, information is retrieved for databases in RAM; if this is FALSE, information is retrieved for databases in ROM.

↔ *pList*
An array of objects, each of the [CDbList](#) class. The caller must preallocate these objects. Upon return, each object receives information about a database on the handheld.

↔ *rCnt*
On entry, the number of objects that the caller allocated in *pList*. Upon return, the actual number of contiguous objects that were filled in by this function.

Returns If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following error code values:

SYNCERR_COMM_NOT_INIT

SYNCERR_LOST_CONNECTION

SYNCERR_REMOTE_SYS

SYNCERR_REMOTE_MEM

Classic Sync Manager API

SyncReadDBList

SYNCERR_REMOTE_BAD_ARG

SYNCERR_NOT_FOUND

There are no more databases to list.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments

To use this function, allocate an array of [CDBList](#) objects. In the *rCnt* parameter, you must specify the number of entries that you allocated in the array. Upon return, this value is updated with the actual number of array entries that were filled in by this function.

Note that the Sync Manager can optimize this transaction if you specify a large number of CDBList objects. You can call the [SyncReadSingleCardInfo\(\)](#) function to determine the number of databases and then allocate that many objects. If you cannot use this strategy, 40 objects is a reasonable intermediate array size.

To iterate through all of the databases in RAM or ROM on the handheld, make a series of calls to *SyncReadDBList()* in this way:

- Set *startIx* to 0 for the first call, and subsequently increment it by the number of entries retrieved by the previous call.
- Reset *rCnt* to the number of entries in your array before each call.

This function is slow when communicating with a handheld that is running a version of Palm OS earlier than 3.0, because those versions of Palm OS return only one entry at a time.

Compatibility

Sync Manager version: 2.0 or later.

Palm OS version: All.

See Also

[CDBList](#), [SyncReadSingleCardInfo\(\)](#)

SyncReadDBSortInfoBlock Function

Purpose Retrieves the sort info block, if one exists, from a classic database on the handheld.

Declared In SyncMgr.h

Prototype SInt32 SyncReadDBSortInfoBlock (HSByte *fHandle*,
CDBGenInfo &*rInfo*)

Parameters → *fHandle*
A handle to a classic record or resource database on the handheld. This handle is returned by a call to [SyncOpenDB\(\)](#) or [SyncCreateDB\(\)](#).

↔ *rInfo*
An object of the [CDBGenInfo](#) class, which specifies and receives information about the sort info block and database.

Returns If successful, returns SYNCERR_NONE. Note that SyncReadDBSortInfoBlock() returns SYNCERR_NONE even if the buffer you allocated was too small for the resource, with version-specific details as follows:

- if you are using version 2.1 or later of the Sync Manager API, the first *m_TotalBytes* of block data is copied to your buffer, and the *m_BytesRead* field of *rInfo* is set to the actual block size.
- if you are using a version of the Sync Manager API earlier than version 2.1, nothing is copied to your buffer, but the *m_BytesRead* field of *rInfo* is set to the actual block size.

Because the Sync Manager does not generate an error for this condition, you must test for it upon function return with an error code of 0: if *m_BytesRead* is greater than *m_TotalBytes*, the buffer was too small.

If unsuccessful, returns one of the following error code values:

SYNCERR_COMM_NOT_INIT
SYNCERR_LOST_CONNECTION
SYNCERR_REMOTE_SYS
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_BAD_ARG

Classic Sync Manager API

SyncReadDBSortInfoBlock

SYNCERR_NOT_FOUND

The requested sort info block is not available.

SYNCERR_NO_FILES_OPEN

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments

This function reads the sort info block from a classic database that is open for reading.

To use this function, allocate the `m_pBytes` buffer in an object of the [`CDbGenInfo`](#) class and fill in the `m_TotalBytes` member of the object. This function fills in the remaining members of the object.

For performance optimization information, see “[Sync Manager and Performance](#)” on page 31 in the *C/C++ Sync Suite Companion*.

Compatibility

Sync Manager version: 2.0 or later.

Palm OS version: All.

See the “Result” section for version-based behavior differences.

See Also

[`CDbGenInfo`](#), [`SyncOpenDB\(\)`](#), [`SyncCreateDB\(\)`](#),
[`SyncWriteDBSortInfoBlock\(\)`](#),
[`SyncReadDBAppInfoBlock\(\)`](#)

SyncReadNextModifiedRec Function

Purpose Retrieves the next modified, archived, or deleted record from a classic record database on the handheld. Each call to this iterator function retrieves the next modified record until all modified records have been returned.

Declared In SyncMgr.h

Prototype SInt32 SyncReadNextModifiedRec
(CRawRecordInfo &rInfo)

Parameters ↔ rInfo

An object of the [CRawRecordInfo](#) class, which specifies and receives information about the record and database. The caller must specify the m_FileHandle, m_RecId, and m_TotalBytes members.

Returns If successful, returns SYNCERR_NONE. Note that this function returns SYNCERR_NONE even if the buffer you allocated was too small for the record, with version-specific details as follows:

- If you are using version 2.1 or later of the Sync Manager API, the first m_TotalBytes of record data is copied to your buffer, and the m_RecId, m_Attribs, m_CatId, m_RecIndex, and m_RecSize fields of rInfo are filled in correctly.
- If you are using a version of the Sync Manager API earlier than version 2.1, nothing is copied to your buffer, but the m_RecId, m_Attribs, m_CatId, and m_RecSize fields of rInfo are filled in correctly.

Because the Sync Manager does not generate an error for this condition, you must test for it upon function return with an error code of 0: if m_RecSize is greater than m_TotalBytes, the buffer was too small.

If unsuccessful, returns one of the following error code values:

SYNCERR_COMM_NOT_INIT
SYNCERR_LOST_CONNECTION
SYNCERR_REMOTE_SYS
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_BAD_ARG

Classic Sync Manager API

SyncReadNextModifiedRec

SYNCERR_NOT_FOUND

There are no more modified records to retrieve.

SYNCERR_REMOTE_BUSY

SYNCERR_NO_FILES_OPEN

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments

To use this function, allocate the `m_pBytes` buffer in an object of the [`CRawRecordInfo`](#) class and fill in the `m_TotalBytes` and `m_FileHandle` members of the object. You must fill in `m_TotalBytes` with the size of the buffer that you allocated and assigned to `m_pBytes`. Note that the database must be open for reading before calling this function.

This function retrieves the record, stores it into the buffer, and fills in the `m_RecId`, `m_Attribs`, `m_CatId`, `m_RecSize`, and `m_RecIndex` fields in `rInfo`.

For general information about iterating through a handheld database, see “[Iterating Through a Database](#)” on page 38 in the *C/C++ Sync Suite Companion*.

You need to be aware of possible difficulties with modifying a database while iterating through it. For more information, see “[Modifying a Database While Iterating](#)” on page 39 in the *C/C++ Sync Suite Companion*.

For performance optimization information, see “[Sync Manager and Performance](#)” on page 31.

Compatibility

Sync Manager version: 2.0 or later.

Palm OS version: All.

See the “Result” section for version-based behavior differences.

See Also

[`CRawRecordInfo`](#),
[`SyncReadNextModifiedRecInCategory\(\)`](#),
[`SyncReadNextRecInCategory\(\)`](#), [`SyncResetRecordIndex\(\)`](#)

SyncReadNextModifiedRecInCategory Function

Purpose Retrieves the next modified record, including deleted and archived records, in a category from a classic database on the handheld. Each call to this iterator function retrieves the next modified record in a category, until all modified records have been retrieved.

Declared In SyncMgr.h

Prototype SInt32 SyncReadNextModifiedRecInCategory
(CRawRecordInfo &rInfo)

Parameters $\leftrightarrow rInfo$
An object of the [CRawRecordInfo](#) class, which specifies and receives information about the record, category, and database. The caller must specify the `m_FileHandle`, `m_CatId`, `m_RecId`, and `m_TotalBytes` members.

Returns If successful, returns SYNCERR_NONE. Note that this function returns 0 even if the buffer you allocated was too small for the record, with version-specific details as follows:

- if you are using version 2.1 or later of the Sync Manager API, the first `m_TotalBytes` of record data is copied to your buffer, and the `m_RecId`, `m_Attribs`, `m_RecIndex`, and `m_RecSize` fields of `rInfo` are filled in correctly.
- if you are using a version of the Sync Manager API earlier than version 2.1, nothing is copied to your buffer, but the `m_RecId`, `m_Attribs`, and `m_RecSize` fields of `rInfo` are filled in correctly.

Because the Sync Manager does not generate an error for this condition, you must test for it upon function return with an error code of 0: if `m_RecSize` is greater than `m_TotalBytes`, the buffer was too small.

If unsuccessful, returns one of the following error code values:

SYNCERR_COMM_NOT_INIT
SYNCERR_LOST_CONNECTION
SYNCERR_REMOTE_SYS
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_BAD_ARG

Classic Sync Manager API

SyncReadNextModifiedRecInCategory

SYNCERR_NOT_FOUND

There are no more modified records to retrieve.

SYNCERR_RECORD_BUSY

SYNCERR_NO_FILES_OPEN

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments

To use this function, allocate the `m_pBytes` buffer an object of the [`CRawRecordInfo`](#) class and fill in the `m_CatId`, `m_TotalBytes` and `m_FileHandle` members of the object. You must fill in `m_TotalBytes` with the size of the buffer that you allocated and assigned to `m_pBytes`. Note that the database must be open for reading before calling this function.

This function retrieves the record, stores it into the buffer, and fills in the `m_RecId`, `m_Attribs`, `m_RecSize`, and `m_RecIndex` fields of `rInfo`.

For general information about iterating through a handheld database, see “[Iterating Through a Database](#)” on page 38 in the *C/C++ Sync Suite Companion*.

You need to be aware of possible difficulties with modifying a database while iterating through it. For more information, see “[Modifying a Database While Iterating](#)” on page 39 in the *C/C++ Sync Suite Companion*.

For performance optimization information, see “[Sync Manager and Performance](#)” on page 31.

Compatibility

Sync Manager version: 2.0 or later.

Palm OS version: All.

See the “Result” section for version-based behavior differences.

See Also

[`CRawRecordInfo`](#), [`SyncReadNextModifiedRec\(\)`](#),
[`SyncReadNextRecInCategory\(\)`](#), [`SyncResetRecordIndex\(\)`](#)

SyncReadNextRecInCategory Function

Purpose Retrieves any record, including deleted, archived, and modified records, in a category from a classic record database on the handheld. Each call to this iterator function retrieves the next record in a category, until all records have been retrieved.

Declared In SyncMgr.h

Prototype SInt32 SyncReadNextRecInCategory
(CRawRecordInfo &rInfo)

Parameters $\leftrightarrow rInfo$
An object of the [CRawRecordInfo](#) class, which specifies and receives information about the record, category, and database. The caller must specify the m_FileHandle, m_CatId, m_RecId, and m_TotalBytes members.

Returns If successful, returns SYNCERR_NONE. Note that this function returns 0 even if the buffer you allocated was too small for the record, with version-specific details as follows:

- if you are using version 2.1 or later of the Sync Manager API, the first m_TotalBytes of record data is copied to your buffer, and the m_RecId, m_Attribs, m_RecIndex, and m_RecSize fields of rInfo are filled in correctly.
- if you are using a version of the Sync Manager API earlier than version 2.1, nothing is copied to your buffer, but the m_RecId, m_Attribs, and m_RecSize fields of rInfo are filled in correctly.

Because the Sync Manager does not generate an error for this condition, you must test for it upon function return with an error code of 0: if m_RecSize is greater than m_TotalBytes, the buffer was too small.

If unsuccessful, returns one of the following error code values:

SYNCERR_COMM_NOT_INIT
SYNCERR_LOST_CONNECTION
SYNCERR_REMOTE_SYS
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_BAD_ARG

Classic Sync Manager API

SyncReadNextRecInCategory

SYNCERR_NOT_FOUND

There are no more modified records to retrieve.

SYNCERR_RECORD_BUSY

SYNCERR_NO_FILES_OPEN

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments

To use this function, allocate the `m_pBytes` buffer an object of the [`CRawRecordInfo`](#) class and fill in the `m_CatId`, `m_TotalBytes` and `m_FileHandle` members of the object. You must fill in `m_TotalBytes` with the size of the buffer that you allocated and assigned to `m_pBytes`. Note that the database must be open for reading before calling this function.

This function retrieves the record, stores it into the buffer, and fills in the `m_RecId`, `m_Attribs`, `m_RecSize`, and `m_RecIndex` fields of `rInfo`.

For general information about iterating through a handheld database, see “[Iterating Through a Database](#)” on page 38 in the *C/C++ Sync Suite Companion*.

You need to be aware of possible difficulties with modifying a database while iterating through it. For more information, see “[Modifying a Database While Iterating](#)” on page 39 in the *C/C++ Sync Suite Companion*.

For performance optimization information, see “[Sync Manager and Performance](#)” on page 31.

Compatibility

Sync Manager version: 2.0 or later.

Palm OS version: All.

See the “Result” section for version-based behavior differences.

See Also

[`CRawRecordInfo`](#), [`SyncReadNextModifiedRec\(\)`](#),
[`SyncReadNextModifiedRecInCategory\(\)`](#),
[`SyncResetRecordIndex\(\)`](#)

SyncReadOpenDbInfo Function

Purpose	Retrieves comprehensive information about a classic database on the handheld.
Declared In	SyncMgr.h
Prototype	<pre>SInt32 SyncReadOpenDbInfo (SyncReadOpenDbInfoParams &rParam, SyncDatabaseInfoType &rInfo)</pre>
Parameters	<p>→ <i>rParam</i> A pointer to a structure of type SyncReadOpenDbInfoParams that specifies the database handle and options for how the database was opened.</p> <p>← <i>rInfo</i> A pointer to a structure of type SyncDatabaseInfoType that receives information about the database.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_MEM SYNCERR_REMOTE_BAD_ARG SYNCERR_NO_FILES_OPEN</p> <p>See “Common Sync Manager Error Codes” on page 486 for a description of all Sync Manager error codes.</p>
Comments	Ensure that the database is already open for reading before calling this function. To use <code>SyncReadOpenDbInfo()</code> , fill in a <code>SyncReadOpenDbInfoParams</code> structure with the database handle and flags that specify how the database was opened. This function fills in a <code>SyncDatabaseInfoType</code> structure with information about the database.

Classic Sync Manager API

SyncReadOpenDbInfo

Compatibility Sync Manager version: 2.2 or later.
Palm OS version: 3.0 or later.

See Also [SyncReadOpenDbInfoParams](#), [SyncDatabaseInfoType](#),
[SyncFindDbByName\(\)](#), [SyncFindDbByTypeCreator\(\)](#)

SyncReadPositionXMap Function

Purpose	Retrieves a list of the record IDs in their sorted order from a classic database on the handheld.
Declared In	<code>SyncMgr.h</code>
Prototype	<code>SInt32 SyncReadPositionXMap (CPositionInfo &rInfo)</code>
Parameters	<code>↔ rInfo</code> An object of the CPositionInfo class, which specifies the database, where to begin reading, and receives the record IDs in sorted order. The caller must specify the <code>m_FileHandle</code> , <code>m_FirstPos</code> , <code>m_MaxEntries</code> , and <code>m_TotalBytes</code> members.
Returns	If successful, returns <code>SYNCERR_NONE</code> . Note that this function returns <code>SYNCERR_NONE</code> if the starting index, <code>m_FirstPos</code> , is out of bounds, in which case <code>m_NumReadIn</code> is set to 0. If unsuccessful, returns one of the following error code values: <code>SYNCERR_COMM_NOT_INIT</code> <code>SYNCERR_FILE_NOT_OPEN</code> <code>SYNCERR_LOCAL_BUFF_TOO_SMALL</code> The value specified by <code>rInfo.m_MaxEntries</code> is greater than 0x3FFF entries. (Because <code>m_TotalBytes</code> is a <code>UInt16</code> , the buffer is limited to 64 KB, which is the size of 16 K four-byte record IDs.) <code>SYNCERR_LOST_CONNECTION</code> <code>SYNCERR_REMOTE_SYS</code> <code>SYNCERR_REMOTE_BAD_ARG</code> <code>SYNCERR_NOT_FOUND</code> <code>SYNCERR_NO_FILES_OPEN</code> See “ Common Sync Manager Error Codes ” on page 486 for a description of all Sync Manager error codes.
Comments	Conduits can use this function to apply the handheld’s record order to the database on the desktop computer. However, this approach might be slower than simply sorting the desktop records using the handheld sort criteria.

Classic Sync Manager API

SyncReadPositionXMap

To use this function, allocate the `m_pBytes` buffer in an object of the [CPositionInfo](#) class and fill in the `m_FirstPos`, `m_MaxEntries`, `m_TotalBytes` and `m_FileHandle` members of the object. The value of `m_TotalBytes` must be the size of the buffer that you allocated and assigned to `m_pBytes`. Note that the database must be open for reading before calling this function.

This function retrieves the record IDs, stores them into the buffer, and fills in the `m_NumReadIn` field of `rInfo`.

`SyncReadPositionXMap()` does not convert the retrieved record IDs to your desktop computer's byte-ordering convention. Each 4-byte record ID is returned in big-endian (Motorola) byte ordering, with the most significant byte stored in the lower memory address. This differs from the behavior of other Sync Manager record-reading functions, which do convert the ID to desktop computer's byte ordering. This means that when you compare record IDs returned by `SyncReadPositionXMap()` with record IDs returned by other Sync Manager functions, you must first convert the former to the desktop computer's format.

Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.
----------------------	--

Sync Manager Version 2.5

This function fails and returns `SYNCERR_LOCAL_BUFF_TOO_SMALL` if the value you specify for `rInfo.m_MaxEntries` is greater than `0x3FFF` entries. This check is not performed in earlier versions.

Sync Manager Version 2.4

A problem is fixed with this function in which it did not pass back a correct value for the number of records passed back (`rInfo.m_NumReadIn`) when the start index (`rInfo.m_FirstPos`) is nonzero. In this version, it correctly passes back the number of record IDs in the `m_pBytes` array.

Sync Manager Version 2.2

A problem is fixed in which Sync Manager crashes if you request that it return a subset of the record IDs. This problem is fixed in Sync Manager versions 2.2 and later.

You can easily get around this problem in earlier versions of the Sync Manager API by having `SyncReadPositionXMap()` retrieve all the record IDs at once. To do so, follow these steps:

1. Retrieve the count of records in the database by calling the [`SyncGetDBRecordCount\(\)`](#) function.
2. Allocate the `m_pBytes` buffer to accommodate that many record IDs. Compute the required size as follows:
`bufsize = count * sizeof(DWORD)`
3. Call `SyncReadPositionXMap()` to retrieve all the record IDs.

See Also [`CPositionInfo`](#)

Classic Sync Manager API

SyncReadRecordById

SyncReadRecordById Function

Purpose	Retrieves a record, by unique record ID, from a classic record database on the handheld.
Declared In	SyncMgr.h
Prototype	SInt32 SyncReadRecordById (CRawRecordInfo & <i>rInfo</i>)
Parameters	<i>↔ rInfo</i> An object of the CRawRecordInfo class, which specifies and receives information about the record and database. The caller must specify the <i>m_FileHandle</i> , <i>m_RecId</i> , and <i>m_TotalBytes</i> members.
Returns	If successful, returns SYNCERR_NONE. Note that this function returns SYNCERR_NONE even if the buffer you allocated was too small for the record, with version-specific details as follows: <ul style="list-style-type: none">if you are using version 2.1 or later of the Sync Manager API, the first <i>m_TotalBytes</i> of record data is copied to your buffer, and the <i>m_RecId</i>, <i>m_Attribs</i>, <i>m_CatId</i>, <i>m_RecIndex</i>, and <i>m_RecSize</i> fields of <i>rInfo</i> are filled in correctly.if you are using a version of the Sync Manager API earlier than version 2.1, nothing is copied to your buffer, but the <i>m_RecId</i>, <i>m_Attribs</i>, <i>m_CatId</i>, and <i>m_RecSize</i> fields of <i>rInfo</i> are filled in correctly. Because the Sync Manager does not generate an error for this condition, you must test for it upon function return with an error code of 0: if <i>m_RecSize</i> is greater than <i>m_TotalBytes</i> , the buffer was too small. If unsuccessful, returns one of the following error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_MEM SYNCERR_REMOTE_BAD_ARG SYNCERR_NOT_FOUND SYNCERR_RECORD_BUSY

SYNCERR_NO_FILES_OPEN

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments

To use this function, allocate the `m_pBytes` buffer in an object of the [CRawRecordInfo](#) class, and fill in the `m_RecId`, `m_TotalBytes` and `m_FileHandle` members of the object. You must fill in `m_TotalBytes` with the size of the buffer that you allocated and assigned to `m_pBytes`. Note that the database must be open for reading before calling this function.

This function retrieves the record, stores it into the buffer, and fills in the `m_RecIndex`, `m_CatId`, `m_Attribs`, and `m_RecSize` fields of `rInfo`.

You need to be aware of possible difficulties with modifying a database while iterating through it. For more information, see “[Modifying a Database While Iterating](#)” on page 39 in the C/C++ Sync Suite Companion.

For performance optimization information, see “[Sync Manager and Performance](#)” on page 31 in the C/C++ Sync Suite Companion.

Compatibility

Sync Manager version: 2.0 or later.

Palm OS version: All.

See the “Result” section for version-based behavior differences.

See Also

[CRawRecordInfo](#), [SyncReadRecordByIndex\(\)](#),
[SyncReadNextModifiedRec\(\)](#),
[SyncReadNextModifiedRecInCategory\(\)](#),
[SyncReadNextRecInCategory\(\)](#)

Classic Sync Manager API

SyncReadRecordByIndex

SyncReadRecordByIndex Function

Purpose	Retrieves a record, by index, from a classic record database on the handheld.
Declared In	SyncMgr.h
Prototype	SInt32 SyncReadRecordByIndex (CRawRecordInfo & <i>rInfo</i>)
Parameters	<i>↔ rInfo</i> An object of the CRawRecordInfo class, which specifies and receives information about the record and database. The caller must specify the <i>m_FileHandle</i> , <i>m_RecIndex</i> , and <i>m_TotalBytes</i> members.
Returns	If successful, returns SYNCERR_NONE. Note that this function returns SYNCERR_NONE even if the buffer you allocated was too small for the record, with version-specific details as follows: <ul style="list-style-type: none">if you are using version 2.1 or later of the Sync Manager API, the first <i>m_TotalBytes</i> of record data is copied to your buffer, and the <i>m_RecId</i>, <i>m_Attribs</i>, <i>m_CatId</i>, <i>m_RecIndex</i>, and <i>m_RecSize</i> fields of <i>rInfo</i> are filled in correctly.if you are using a version of the Sync Manager API earlier than version 2.1, nothing is copied to your buffer, but the <i>m_RecId</i>, <i>m_Attribs</i>, <i>m_CatId</i>, and <i>m_RecSize</i> fields of <i>rInfo</i> are filled in correctly.

Because the Sync Manager does not generate an error for this condition, you must test for it upon function return with an error code of 0: if *m_RecSize* is greater than *m_TotalBytes*, the buffer was too small.

If unsuccessful, returns one of the following error code values:

SYNCERR_COMM_NOT_INIT
SYNCERR_LOST_CONNECTION
SYNCERR_REMOTE_SYS
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_BAD_ARG
SYNCERR_NOT_FOUND

SYNCERR_RECORD_BUSY

SYNCERR_NO_FILES_OPEN

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments

To use this function, allocate the `m_pBytes` buffer in an object of the [CRawRecordInfo](#) class and fill in the `m_RecIndex`, `m_TotalBytes` and `m_FileHandle` members of the object. You must fill in `m_TotalBytes` with the size of the buffer that you allocated and assigned to `m_pBytes`. Note that the database must be open for reading before calling this function.

This function retrieves the record, stores it into the buffer, and fills in the `m_RecId`, `m_CatId`, `m_Attribs`, and `m_RecSize` fields of `rInfo`.

You need to be aware of possible difficulties with modifying a database while iterating through it. For more information, see “[Modifying a Database While Iterating](#)” on page 39 in the C/C++ Sync Suite Companion.

For performance optimization information, see “[Sync Manager and Performance](#)” on page 31 in the C/C++ Sync Suite Companion.

Compatibility

Sync Manager version: 2.0 or later.

Palm OS version: All.

See the “Result” section for version-based behavior differences.

See Also

[CRawRecordInfo](#), [SyncReadRecordById\(\)](#),
[SyncReadNextModifiedRec\(\)](#),
[SyncReadNextModifiedRecInCategory\(\)](#),
[SyncReadNextRecInCategory\(\)](#)

Classic Sync Manager API

SyncReadResRecordByIndex

SyncReadResRecordByIndex Function

Purpose	Retrieves a resource record, by index, from an open resource database on the handheld.
Declared In	SyncMgr.h
Prototype	SInt32 SyncReadResRecordByIndex (CRawRecordInfo & <i>rInfo</i> , SInt32 <i>bBody</i> =TRUE)
Parameters	<p>↔ <i>rInfo</i> An object of the CRawRecordInfo class, which specifies and receives information about the record and database. The caller must specify the <i>m_FileHandle</i>, <i>m_RecIndex</i>, and <i>m_TotalBytes</i> members.</p> <p>→ <i>bBody</i> A Boolean value. If this is TRUE, the resource data is retrieved and stored into the <i>m_pBytes</i> buffer in <i>rInfo</i>. If not, the other fields in <i>rInfo</i> are filled in, but the resource data is not copied.</p>
Returns	<p>If successful, returns SYNCERR_NONE. Note that this function returns SYNCERR_NONE even if the buffer you allocated was too small for the resource, with version-specific details as follows:</p> <ul style="list-style-type: none">if you are using version 2.1 or later of the Sync Manager API, the first <i>m_TotalBytes</i> of resource data is copied to your buffer, and the <i>m_RecId</i>, <i>m_CatId</i>, <i>m_RecIndex</i>, and <i>m_RecSize</i> fields of <i>rInfo</i> are filled in correctly.if you are using a version of the Sync Manager API earlier than version 2.1, nothing is copied to your buffer, but the <i>m_RecId</i>, <i>m_CatId</i>, and <i>m_RecSize</i> fields of <i>rInfo</i> are filled in correctly. <p>Because the Sync Manager does not generate an error for this condition, you must test for it upon function return with an error code of 0: if <i>m_RecSize</i> is greater than <i>m_TotalBytes</i>, the buffer was too small.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS</p>

SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_BAD_ARG
SYNCERR_NOT_FOUND
SYNCERR_NO_FILES_OPEN

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments

To use this function, allocate the `m_pBytes` buffer in object of the [`CRawRecordInfo`](#) class and fill in the `m_RecIndex`, `m_TotalBytes` and `m_FileHandle` members of the object. You must fill in `m_TotalBytes` with the size of the buffer that you allocated and assigned to `m_pBytes`. Note that the database must be open for reading before calling this function.

This function retrieves the resource, stores it into the buffer, and fills in the `m_RecId`, `m_CatId`, and `m_RecSize` fields of `rInfo`.

To start iterating through a resource database, you can set `m_RecIndex` to 0 and call this function. You can retrieve subsequent resources by incrementing the previous value of `m_RecIndex` by 1. Continue calling this function until an error code is returned. You must be sure to “refresh” the value of `m_RecIndex`, because this function overloads that field by returning the resource ID in it.

For performance optimization information, see “[Sync Manager and Performance](#)” on page 31 in the *C/C++ Sync Suite Companion*.

Compatibility

Sync Manager version: 2.0 or later.

Palm OS version: All.

See the “Result” section for version-based behavior differences.

See Also

[`CRawRecordInfo`](#), [`SyncWriteResourceRec\(\)`](#)

Classic Sync Manager API

SyncResetRecordIndex

SyncResetRecordIndex Function

Purpose Resets the record iteration index of an open, classic record database on the handheld.

Declared In SyncMgr.h

Prototype SInt32 SyncResetRecordIndex (HSByte *fHandle*)

Parameters → *fHandle*

A handle to an open record database on the handheld. This handle is returned by a call to the [SyncOpenDB\(\)](#) or [SyncCreateDB\(\)](#) functions.

Returns If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following error code values:

SYNCERR_COMM_NOT_INIT

SYNCERR_LOST_CONNECTION

SYNCERR_REMOTE_BAD_ARG

SYNCERR_NO_FILES_OPEN

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments Call this function before iterating through a database with functions such as `SyncReadNextRecInCategory()` or `SyncReadNextModifiedRecInCategory()`. You do not need to call this function before iterating through a database for the first time; however, if your conduit iterates through the same database more than once while you have it open, you need to call this function to reset the index before beginning the second (or subsequent) iteration.

For general information about iterating through a handheld database, see “[Iterating Through a Database](#)” on page 38 in the *C/C++ Sync Suite Companion*.

You need to be aware of possible difficulties with modifying a database while iterating through it. For more information, see “[Modifying a Database While Iterating](#)” on page 39 in the *C/C++ Sync Suite Companion*.

Compatibility Sync Manager version: 2.0 or later.
Palm OS version: All.

See Also [SyncReadNextModifiedRec\(\)](#),
[SyncReadNextModifiedRecInCategory\(\)](#),
[SyncReadNextRecInCategory\(\)](#), [SyncOpenDB\(\)](#),
[SyncCreateDB\(\)](#)

Classic Sync Manager API

SyncResetSyncFlags

SyncResetSyncFlags Function

Purpose Resets the modified flag of all records in the open, classic record database on the handheld. Resets the backup date for an open, classic record or resource database.

Declared In SyncMgr.h

Prototype SInt32 SyncResetSyncFlags (HSByte *fHandle*)

Parameters → *fHandle*

A handle to the database on the handheld. This handle is returned by a call to the [SyncOpenDB\(\)](#) or [SyncCreateDB\(\)](#) functions.

Returns If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following error code values:

SYNCERR_COMM_NOT_INIT

SYNCERR_LOST_CONNECTION

SYNCERR_REMOTE_SYS

SYNCERR_REMOTE_BAD_ARG

SYNCERR_NO_FILES_OPEN

SYNCERR_READ_ONLY

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments If *fHandle* is a handle to a record database, the database must be open for reading and writing. If *fHandle* is a handle to a resource database, the database can be open for either read-only or read-write access.

For more information about the record flags, see “[eSyncRecAttrs](#)” on page 425.

NOTE: SyncResetSyncFlags () works only for classic databases. To reset the modification state for a schema database, see the description of [SyncDbCloseDatabase\(\)](#) instead.

Compatibility Sync Manager version: 2.0 or later.
Palm OS version: All.

See Also [eSyncRecAttrs](#), [SyncOpenDB\(\)](#), [SyncCreateDB\(\)](#)

Classic Sync Manager API

SyncWriteAppPreference

SyncWriteAppPreference Function

Purpose	Writes application preferences into a preferences database on the handheld.
Declared In	SyncMgr.h
Prototype	SInt32 SyncWriteAppPreference (CRawPreferenceInfo &rInfo)
Parameters	$\leftrightarrow rInfo$ An object of the CRawPreferenceInfo class, which specifies and receives information about the application preferences.
Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_MEM SYNCERR_REMOTE_BAD_ARG SYNCERR_UNKNOWN_REQUEST See " Common Sync Manager Error Codes " on page 486 for a description of all Sync Manager error codes.
Comments	To use this function, allocate the m_pBytes buffer in an object of the CRawPreferenceInfo class, store the preference information into the buffer, and fill in the following other members in the object: m_version, m_creator, m_prefId, m_backedUp, and m_nBytes. The structure of the data in the preferences block is application dependent. The Sync Manager does not modify this data in any way when sending it. This means that multi-byte integer data is stored using big-endian byte ordering, with the most significant byte stored at the lower address in memory. Your conduit is responsible for performing any necessary byte swapping. You must be sure that the version and size of the preference block is compatible with the version of the application running on the handheld. Writing an improperly sized preference block can corrupt a database.

IMPORTANT: You must be sure that the version and size of the preference block is compatible with the version of the application running on the handheld. Writing an improperly sized preference block can corrupt a database.

Compatibility

Sync Manager version: 2.0 or later.

Palm OS version: 2.0 or later.

Application preferences can be no larger than 64 KB, in Palm OS Cobalt as well as in Palm OS Garnet and earlier releases. Therefore you can use this function to write preferences to handhelds running any of the above versions of Palm OS.

See Also

[CRawPreferenceInfo](#), [SyncReadAppPreference \(\)](#)

Classic Sync Manager API

SyncWriteDBAppInfoBlock

SyncWriteDBAppInfoBlock Function

Purpose	Writes an application info block to a classic record or resource database on the handheld.
Declared In	SyncMgr.h
Prototype	SInt32 SyncWriteDBAppInfoBlock (HSByte <i>fHandle</i> , CDBGenInfo & <i>rInfo</i>)
Parameters	<p>→ <i>fHandle</i> A handle to a classic record or resource database on the handheld. This handle is returned by a call to SyncOpenDB() or SyncCreateDB().</p> <p>→ <i>rInfo</i> An object of the CDBGenInfo class, which specifies information about the application info block and database.</p>
Returns	If successful, returns SYNCERR_NONE, which means that the block was written to the handheld database. If unsuccessful, returns one of the following error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_MEM SYNCERR_REMOTE_BAD_ARG SYNCERR_READ_ONLY SYNCERR_NO_FILES_OPEN See “ Common Sync Manager Error Codes ” on page 486 for a description of all Sync Manager error codes.

Comments This function writes an application information block to a classic database on the handheld. The database must be opened for reading and writing. For more information about application information blocks, see “[Categories and the Application Info Block](#)” on page 145 in the *Introduction to Conduit Development*.

To use this function, allocate the `m_pBytes` buffer in an object of the [`CDbGenInfo`](#) class and store the information into the buffer. Then set the `m_TotalBytes` and `m_BytesRead` members to the size of the data buffer.

NOTE: Because of a problem in earlier versions of the Sync Manager API, you must assign the size of the information block to both the `m_BytesRead` and `m_TotalBytes` fields of your `CDbGenInfo` object before calling `SyncWriteDbAppInfoBlock()`.

To completely delete the application info block from the handheld database, set both `m_TotalBytes` and `m_BytesRead` to 0.

For performance optimization information, see “[Sync Manager and Performance](#)” on page 31.

Compatibility Sync Manager version: 2.0 or later.
Palm OS version: All.

See Also [`CDbGenInfo`](#), [`SyncReadDBAppInfoBlock\(\)`](#)

Classic Sync Manager API

SyncWriteDBSortInfoBlock

SyncWriteDBSortInfoBlock Function

Purpose	Writes a sort information block to an classic record or resource database on the handheld.
Declared In	SyncMgr.h
Prototype	SInt32 SyncWriteDBSortInfoBlock (HSByte <i>fHandle</i> , CDbGenInfo & <i>rInfo</i>)
Parameters	<p>→ <i>fHandle</i> A handle to a classic record or resource database on the handheld. This handle is returned by a call to SyncOpenDB() or SyncCreateDB(). The database must be open for reading and writing.</p> <p>→ <i>rInfo</i> An object of the CDbGenInfo class, which specifies information about the sort info block and database.</p>
Returns	If successful, returns SYNCERR_NONE, which means the block was written to the handheld database. If unsuccessful, returns one of the following error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_MEM SYNCERR_REMOTE_BAD_ARG SYNCERR_READ_ONLY SYNCERR_NO_FILES_OPEN See “ Common Sync Manager Error Codes ” on page 486 for a description of all Sync Manager error codes.

Comments This function writes a sort info block to a classic database on the handheld that is open for reading and writing.

To use this function, allocate the `m_pBytes` buffer in an object of the [CDBGenInfo](#) class and store the sorting information in that buffer. Then assign the size of your buffer to the `m_TotalBytes` field.

To completely delete the sort info block from the handheld database, set `m_TotalBytes` to 0.

For performance optimization information, see “[Sync Manager and Performance](#)” on page 31.

Compatibility Sync Manager version: 2.0 or later.
Palm OS version: All.

See Also [CDBGenInfo](#), [SyncReadDBSortInfoBlock\(\)](#)

Classic Sync Manager API

SyncWriteRec

SyncWriteRec Function

Purpose	Writes a record to a classic record database on the handheld.
Declared In	SyncMgr.h
Prototype	SInt32 SyncWriteRec (CRawRecordInfo &rInfo)
Parameters	$\leftrightarrow rInfo$ An object of the CRawRecordInfo class, which specifies and receives information about the record and database. The caller must specify the <code>m_FileHandle</code> , <code>m_RecId</code> , <code>m_Attribs</code> , <code>m_CatId</code> , and <code>m_RecSize</code> members.
Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_MEM SYNCERR_REMOTE_BAD_ARG SYNCERR_NO_FILES_OPEN SYNCERR_BAD_OPERATION SYNCERR_READ_ONLY See “ Common Sync Manager Error Codes ” on page 486 for a description of all Sync Manager error codes.
Comments	This function can perform the following write operations on a classic database that is open for writing: <ul style="list-style-type: none">• Overwrite an existing record by specifying its record ID.• Add a new record by supplying 0 as the record ID. Note that the Data Manager on the handheld is responsible for assigning new record IDs. To use this function, allocate the <code>m_pBytes</code> buffer in object of the CRawRecordInfo class and store the record information in that buffer. Then assign the size of your buffer to the <code>m_RecSize</code> field, and fill in the remaining fields with information about the record.

You cannot specify the position in the database of a new record, nor can you rely on a new record being added at the end of the database. Upon completion of synchronization operations, applications on the handheld are sent a `sysAppLaunchCmdSyncNotify` notification, at which time an application can sort or update the database as required.

When you write a record to a handheld that is running a version of Palm OS earlier than version 3.0, the record is always marked as modified. For Palm OS software version 3.0, the record is marked as modified only if the `eRecAttrDirty` flag is set in the `m_Attribs` field of `rInfo`.

If you are synchronizing with a built-in (ROM) database on a handheld running a version of Palm OS earlier than version 2.0, you need to handle a special condition that occurs after a hard reset is performed on the handheld. On these handhelds, the unique record ID seeds generated for record databases are always the same. To avoid record ID collisions, the default conduits zero out existing record IDs on the desktop before restoring the databases on hard-reset handhelds, which forces new, unique record IDs to be generated. You only need to apply this work-around if you are synchronizing with a ROM-based application on a handheld running version 2.0 or earlier of Palm OS.

When you use a version of the Sync Manager API earlier than version 2.0 to write a record to a database on the handheld, the current iteration index is not updated. For more information, see “[Modifying a Database While Iterating](#)” on page 39 in the C/C++ Sync Suite Companion.

Compatibility

Sync Manager version: 2.0 or later.

Palm OS version: All.

See the “Result” section for version-based behavior differences.

See Also

[CRawRecordInfo](#), [SyncReadRecordById\(\)](#),
[SyncReadRecordByIndex\(\)](#)

Classic Sync Manager API

SyncWriteResourceRec

SyncWriteResourceRec Function

Purpose	Writes a resource to a classic resource database on the handheld.
Declared In	SyncMgr.h
Prototype	SInt32 SyncWriteResourceRec (CRawRecordInfo &rInfo)
Parameters	$\rightarrow rInfo$ An object of the CRawRecordInfo class, which specifies information about the resource and database. The caller must specify the m_FileHandle, m_RecId, m_RecIndex, and m_RecSize members.
Returns	If successful, returns SYNCERR_NONE, which means that the resource was written. If unsuccessful, returns one of the following error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_MEM SYNCERR_REMOTE_BAD_ARG SYNCERR_NO_FILES_OPEN SYNCERR_BAD_OPERATION SYNCERR_READ_ONLY See “ Common Sync Manager Error Codes ” on page 486 for a description of all Sync Manager error codes.
Comments	To use this function, allocate the m_pBytes buffer in an object of the CRawRecordInfo class, and store the resource information in that buffer. You then must assign the size of the resource to the m_RecSize field, and fill in m_FileHandle, m_RecId, m_RecIndex. Note that the database must be open for writing before calling this function.
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.
See Also	CRawRecordInfo , SyncReadResRecordByIndex()

Common Sync Manager API

The common Sync Manager API consists of APIs that work with more than one type of database (schema, extended, or classic) or that do not work with databases at all.

This chapter has the following sections:

Common Sync Manager Classes	384
Common Sync Manager Structures and Types	400
Common Sync Manager Constants	405
Common Sync Manager Functions and Macros	434
Common Sync Manager Error Codes	486

All Sync Manager functions are available in Sync20.dll. The API described in this chapter is declared in SyncCommon.h.

Common Sync Manager API

Common Sync Manager Classes

Common Sync Manager Classes

This section describes the following classes, which either schema, extended, classic, or common Sync Manager functions take as parameters. It also includes some classes that are used with the conduit entry point API. These classes define only data members and no methods.

<u>CCallApplicationParams</u>	Defines information that SyncCallDeviceApplication() passes to and receives from an application/module on the handheld.
<u>CCardInfo</u>	Defines information about a memory card on the handheld. Use with SyncReadSingleCardInfo().
<u>CDbList</u>	Receives information about a database on the handheld.
<u>CSyncPreference</u>	Specifies information transferred between HotSync Manager and a conduit when it calls a conduit's ConfigureConduit() entry point.
<u>CSyncProperties</u>	Specifies the properties of the current conduit's synchronization operations when HotSync Manager calls a conduit's OpenConduit() entry point.
<u>CSystemInfo</u>	Defines system information that SyncReadSystemInfo() retrieves from the handheld.
<u>CUserIDInfo</u>	Receives information that SyncReadUserID() passes back about the current handheld user.

CCallApplicationParams

Purpose Defines information that [SyncCallDeviceApplication\(\)](#) passes to and receives from an application on a Palm OS Cobalt handheld.

Declared In SyncCommon.h

Prototype

```
class CCallApplicationParams {
    public:
        char m_dbName[SYNC_DB_NAMELEN];
        UInt32 m_dwCreatorID;
        UInt32 m_dwTypeID;
        UInt16 m_wAttributes;
        UInt16 m_wActionCode;
        UInt32 m_dwParamSize;
        void *m_pParam;
        UInt32 m_dwResultBufSize;
        void *m_pResultBuf;
        UInt32 m_dwresultCode;
        UInt32 m_dwActResultSize;
        UInt32 m_dwReserved;
}
```

Data Members

m_dbName

A TCHAR pointer that specifies the [database name](#) of the target application.

m_dwCreatorID

Specifies the [creator ID](#) of the target application.

m_dwTypeID

Specifies the [database type](#) ID of the target application. Type is used only as a cross-check and may be set to zero if you don't care what the database type is.

m_wAttributes

A combination of dmHdrAttr... bits that specify whether the target application is a classic, extended, or schema database. All other dmHdrAttr bits are ignored. For a description of these bits, see "[Database Attributes](#)" on page 407.

m_wActionCode

The application-specific code that specifies the action to perform.

Common Sync Manager API

CCallApplicationParams

`m_dwParamSize`

Specifies the number of bytes in the `m_pParam` array.

`m_pParam`

Specifies a pointer to the parameter block.

`m_dwResultBufSize`

Specifies the total number of bytes in the `m_pResultBuf` array.

`m_pResultBuf`

Specifies a pointer to the result buffer.

`m_dwresultCode`

Receives the result code returned by the handheld application.

`m_dwActResultSize`

Receives the actual size of the result data.

`m_dwReserved`

Reserved for future use. The caller must set this member to NULL (0) before calling the Sync Manager function.

Comments

This class is similar to [CCallModuleParams](#), but it is used only with [SyncCallDeviceApplication\(\)](#) to enable you to call the target application uniquely identified by creator ID, database name, and database attributes (classic, extended, or schema).

The value of `m_dwActResultSize` is greater than that of `m_dwResultBufSize`, if your buffer is not large enough to accommodate all the results data. In this case, the function copies only `dwResultBufSize` bytes of data into the buffer.

Compatibility

Sync Manager version: 2.5 or later.

Palm OS version: Palm OS Cobalt, version 6.0.1 or later.

See Also

[SyncCallDeviceApplication\(\)](#)

CCardInfo

Purpose Defines information about a memory card on the handheld. Use with [SyncReadSingleCardInfo\(\)](#).

Declared In SyncCommon.h

Prototype

```
class CCardInfo {
    public:
        HSByte m_CardNo;
        UInt16 m_CardVersion;
        SInt32 m_CreateDate;
        UInt32 m_RomSize;
        UInt32 m_RamSize;
        UInt32 m_FreeRam;
        HSByte m_CardNameLen;
        HSByte m_ManufNameLen;
        char m_CardName [SYNC_REMOTE_CARDNAME_BUF_SIZE];
        char
        m_ManufName [SYNC_REMOTE_MANUENAME_BUF_SIZE];
        UInt16 m_romDbCount;
        UInt16 m_ramDbCount;
        UInt32 m_dwReserved;
}
```

Data Members	
m_CardNo	Specifies the memory card number to read.
m_CardVersion	Receives the version of the memory card.
m_CreateDate	Receives the creation date of the memory card. This is a time_t value.
m_RomSize	Receives the amount of ROM on the memory card.
m_RamSize	Receives the total amount of RAM on the memory card.
m_FreeRam	Receives the amount of available RAM on the memory card. See SyncReadSingleCardInfo() for Palm OS version-specific details.
m_CardNameLen	Receives the number of characters in the memory card name.

Common Sync Manager API

CCardInfo

`m_ManufNameLen`

Receives the number of characters in `m_ManufName`.

`m_CardName`

Receives the memory card name in an array of characters.

`m_ManufName`

Receives the manufacturer's name in an array of characters.

`m_romDbCount`

Receives the number of ROM-based databases on the memory card.

`m_ramDbCount`

Receives the number of RAM-based databases on the memory card.

`m_dwReserved`

Reserved for future use. The caller must set this member to NULL (0) before calling the Sync Manager function.

Compatibility

Sync Manager version: All.

Palm OS version: All.

See Also

[SyncReadSingleCardInfo\(\)](#)

CDbList

Purpose	Receives information about a database on the handheld.
Declared In	<code>SyncCommon.h</code>
Prototype	<pre>class CDbList { public: SInt32 m_CardNum; UInt16 m_DbFlags; UInt32 m_DbType; char m_Name[SYNC_DB_NAMELEN]; UInt32 m_Creator; UInt16 m_Version; UInt32 m_ModNumber; UInt16 m_Index; SInt32 m_CreateDate; SInt32 m_ModDate; SInt32 m_BackupDate; SInt32 m_miscFlags; SInt32 m_RecCount; SInt32 m_dwReserved; } typedef class CDbList *CDbListPtr</pre>
Data Members	
m_CardNum	Receives the number of the memory card on which the database is stored. The first memory card in the system is card number 0, and subsequent card numbers are incremented by 1.
m_DbFlags	Receives a combination of eDbFlags values that indicate one or more database characteristics (whether it is a classic record or resource database, an extended database, or a schema database, whether it is to be backed up, and so on).
m_DbType	Receives a 4-byte database type. Palm OS associates special behavior with several identifier values, including 'DATA', 'data', 'appl', 'panl', and 'libr'.
m_Name	Receives the name of the database as a character array of size SYNC_DB_NAMELEN.

Common Sync Manager API

CDbList

`m_Creator`

Receives the creator ID of the database.

`m_Version`

Receives the version of the database.

`m_ModNumber`

Receives the database modification number, which is incremented every time a record in the database is added, modified, or deleted on the handheld.

`m_Index`

Receives the index of the database in the list of databases on the handheld. Note: this value is not set for the

[`SyncReadOpenDbInfo\(\)`](#), [`SyncFindDbByName\(\)`](#), or [`SyncFindDbByTypeCreator\(\)`](#) functions.

`m_CreateDate`

Receives the date on which the database was created. This is a `time_t` value.

`m_ModDate`

Receives the date of the most recent database modification. This is a `time_t` value. Note: versions 1.x of Palm OS did not update the modification date.

`m_BackupDate`

Receives the date of the most recent database backup. This is a `time_t` value.

`m_miscFlags`

Receives a combination of one or more of the [`eMiscDbListFlags`](#) values.

`m_RecCount`

Unused.

`m_dwReserved`

Reserved for future use. The caller must set this member to `NULL` (0) before calling the Sync Manager function.

Compatibility

Sync Manager version: All.

Palm OS version: All.

See Also

[`SyncReadDBList\(\)`](#), [`SyncDatabaseInfoType`](#),
[`SyncReadOpenDbInfo\(\)`](#), [`SyncFindDbByName\(\)`](#),
[`SyncFindDbByTypeCreator\(\)`](#), [`CSyncProperties`](#)

CSyncPreference

Purpose	Specifies information transferred between HotSync Manager and a conduit when it calls a conduit's ConfigureConduit() entry point.
Declared In	<code>SyncCommon.h</code>
Prototype	<pre>class CSyncPreference { public: char m_PathName[BIG_PATH]; char m_Registry[BIG_PATH]; HKEY m_hKey; eSyncPref m_SyncPref; eSyncTypes m_SyncType; UInt32 m_dwReserved; }</pre>
Data Members	<p>m_PathName Specifies the path (as a null-terminated string) to prepend to filenames on the desktop computer.</p> <p>m_Registry Specifies the full Windows registry path (as a null-terminated string) of the current conduit. This member is deprecated; use the Conduit Manager API to access conduit configuration entries instead.</p> <p>m_hKey Specifies the primary Windows registry key for the current conduit. This member is deprecated; use the Conduit Manager API to access conduit configuration entries instead.</p> <p>m_SyncPref Specifies an eSyncPref value for the current conduit to inform HotSync Manager whether the user's selection in the conduit's Custom dialog box is a permanent or temporary setting.</p> <p>m_SyncType Defines an eSyncTypes value that communicates between HotSync Manager and the current conduit the type of selected sync operation to display in the Custom dialog box and what type of sync operation a user then selects. When HotSync Manager calls ConfigureConduit(), it passes the conduit the setting for the next sync operation. When the</p>

Common Sync Manager API

CSyncPreference

conduit returns, it passes HotSync Manager the user's selection.

`m_dwReserved`

Reserved for future use. Set this member to NULL (0).

Compatibility HotSync Manager version: 3.0 or later.
Palm OS version: All.

See Also [ConfigureConduit\(\)](#)

CSyncProperties

Purpose Specifies the properties of the current conduit's synchronization operations when HotSync Manager calls a conduit's [OpenConduit\(\)](#) entry point.

Declared In SyncCommon.h

Prototype

```
class CSyncProperties {
    public:
        eSyncTypes m_SyncType;
        char m_PathName[BIG_PATH];
        char m_LocalName[BIG_PATH];
        char m_UserName[BIG_PATH];
        char *m_RemoteName[SYNC_DB_NAMELEN];
        CDbListPtr *m_RemoteDbList;
        SInt32 m_nRemoteCount;
        UInt32 m_Creator;
        UInt16 m_CardNo;
        UInt32 m_DbType;
        UInt32 m_AppInfoSize;
        UInt32 m_SortInfoSize;
        eFirstSync m_FirstDevice;
        eConnType m_Connection;
        char m_Registry[BIG_PATH];
        HKEY m_hKey;
        UInt32 m_dwReserved;
}
```

Data Members

m_SyncType
Specifies the current synchronization type as one of the [eSyncTypes](#) values. The Sync Manager passes back the eFast or eSlow value based solely on whether this handheld's last HotSync operation was with the current desktop. For non-schema databases, this is sufficient for a conduit to determine whether to perform a fast or slow sync. However, for schema databases, this value is not sufficient. Instead of relying on this value to determine whether to perform a fast or slow sync, conduits that synchronize schema databases must call [SyncDbGetSyncMode\(\)](#). Conduits synchronizing schema databases must still rely on this field for the other values (eHHToPC, ePCToHH, etc.) it receives, which are equally valid for all database types.

Common Sync Manager API

CSyncProperties

`m_PathName`

Specifies the conduit's directory name. This value is set in the conduit's [Directory](#) configuration entry.

`m_LocalName`

Specifies the desktop file that the conduit synchronizes with. This value is set in the conduit's [File](#) configuration entry.

`m_UserName`

Specifies the HotSync user name (as a null-terminated string) of the user who is currently performing a HotSync operation.

`m_RemoteName`

Specifies a pointer to an array of the names of the databases on the handheld that have the conduit's creator ID. Do not rely on data in this array if your conduit is associated with more than 32 databases with the same creator ID on the handheld; use `m_RemoteDbList` instead. See "[Change to Support More than 32 Databases](#)" on page 1120.

`m_RemoteDbList`

Specifies a pointer to an array of [CDbList](#) objects that contain information about the handheld databases that have the same creator ID as the current conduit. The number of items in the array is specified by the `m_nRemoteCount` member.

`m_nRemoteCount`

Specifies the number of entries in the `m_RemoteDbList` array.

`m_Creator`

Specifies the [creator ID](#) associated with the current conduit.

`m_CardNo`

Specifies the card number of the [memory card](#) on the handheld on which databases are stored.

`m_DbType`

Specifies the [database type](#) of the database on the handheld.

`m_AppInfoSize`

Specifies the size of the application info block of the default handheld database. This value is stored in this object for convenience.

`m_SortInfoSize`

Specifies the size of the sort info block of the default handheld database. This value is stored in this object for convenience.

`m_FirstDevice`

Specifies an [eFirstSync](#) value that indicates whether the current HotSync operation is the first for the handheld, the first with the current desktop, or the first for neither.

`m_Connection`

Specifies an [eConnType](#) value that indicates the type transfer medium of the current HotSync operation.

`m_Registry`

Specifies the full Windows registry path (as a null-terminated string) of the current conduit. This member is deprecated; use the [Conduit Manager API](#) to access conduit configuration entries instead.

`m_hKey`

Specifies the primary Windows registry key for the current conduit. This member is deprecated; use the [Conduit Manager API](#) to access conduit configuration entries instead.

`m_dwReserved`

Reserved for future use. Set this member to NULL (0).

Comments

A conduit can use this to determine the general synchronization type of the current HotSync process and the handheld databases it is associated with.

Compatibility

HotSync Manager version: All.

Palm OS version: All.

See “[Change to Support More than 32 Databases](#)” on page 1120.

See Also

[OpenConduit\(\)](#)

Common Sync Manager API

CSystemInfo

CSystemInfo

Purpose Defines system information that [SyncReadSystemInfo\(\)](#) retrieves from the handheld.

Declared In SyncCommon.h

Prototype

```
class CSystemInfo {
    public:
        UInt32 m_RomSoftVersion;
        UInt32 m_LocalId;
        HSByte m_ProdIdLength;
        HSByte m_AllocedLen;
        HSByte *m_ProductIdText;
        UInt32 m_dwReserved;
}
```

Data Members	
<code>m_RomSoftVersion</code>	Receives the Palm OS version from the handheld's ROM during a HotSync operation. The caller can use the SYNCROMVMAJOR() and SYNCROMVMINOR() macros to decode this value into major and minor version numbers. Note that HotSync Manager versions 6.0 and later store this value in the user data store so that it can be accessed outside of a HotSync operation.
<code>m_LocalId</code>	Receives the localization ID for the handheld. Currently, all systems return the value 0x00010000L.
<code>m_ProdIdLength</code>	Receives the actual length of the product ID passed back in <code>m_ProductIdText</code> .
<code>m_AllocedLen</code>	Specifies the number of bytes that the caller allocated for the <code>m_ProductIdText</code> buffer.
<code>m_ProductIdText</code>	Defines an array of bytes to receive the product ID from the handheld. The caller must allocate this buffer with a size of at least <code>SYNC_MAX_PROD_ID_SIZE</code> before calling SyncReadSystemInfo() . This field identifies the type of processor on the handheld. For a list of processor types, see <i>Exploring Palm OS: System Management</i> in the Palm OS SDK.

m_dwReserved

Reserved for future use. The caller must set this member to NULL (0) before calling the Sync Manager function.

Compatibility Sync Manager version: All.
Palm OS version: All.

See Also [SyncReadSystemInfo\(\)](#)

Common Sync Manager API

CUserIDInfo

CUserIDInfo

Purpose Receives information that [SyncReadUserID\(\)](#) passes back about the current handheld user.

Declared In SyncCommon.h

Prototype

```
class CUserIDInfo {
public:
    char m_pName[SYNC_REMOTE_USERNAME_BUF_SIZE];
    SInt32 m_NameLength;
    char m_Password[SYNC_REMOTE_PASSWORD_BUF_SIZE];
    SInt32 m_PasswdLength;
    SInt32 m_LastSyncDate;
    UInt32 m_LastSyncPC;
    UInt32 m_Id;
    UInt32 m_ViewerId;
    UInt32 m_dwReserved;
}
```

Data Members

m_pName

Receives handheld's user name as a null-terminated character array no larger than SYNC_REMOTE_USERNAME_BUF_SIZE. This value is 0 if the handheld has never been synchronized.

m_NameLength

Receives the actual length in bytes of the user name passed back in the m_pName member, including the NULL terminator.

m_Password

Receives the user's password as a null-terminated character array no larger than SYNC_REMOTE_PASSWORD_BUF_SIZE. The password is in an encrypted binary format.

m_PasswdLength

Receives the actual length of the user's encrypted password passed back in m_Password, including the NULL terminator.

m_LastSyncDate

Receives the date of the most recent synchronization of the handheld. This is a time_t value.

This member is set to 0 when [SyncReadUserID\(\)](#) reads information from a handheld running a version of Palm OS

earlier than version 3.0. For versions 3.0 and later, this value is set correctly.

m_LastSyncPC

Receives the PC ID of the desktop computer with which the handheld was most recently synchronized. HotSync Manager generates this value and stores it on the desktop computer when HotSync Manager is installed; it also stores it on the handheld during a HotSync operation. The value in this member is read from the handheld.

m_Id

Receives the handheld's user ID.

m_ViewerId

Receives the ID of the handheld. Not currently used.

m_dwReserved

Reserved for future use. The caller must set this member to NULL (0) before calling the Sync Manager function.

Compatibility Sync Manager version: All.
 Palm OS version: All.

See Also [SyncReadUserID\(\)](#)

Common Sync Manager Structures and Types

This section describes the following structures and types, which the parameters of either schema, extended, classic, or common Sync Manager functions take. It also includes some structures and types that are used with the conduit entry point API.

<u>CONDHANDLE</u>	Defines a handle for registering and unregistering a conduit with Sync Manager.
<u>DBDatabaseInfo</u>	Defines information in a database header. The defined fields depend on whether the database is a schema, extended, or classic database.

CONDHANDLE Typedef

Purpose	Defines a handle for registering and unregistering a conduit with Sync Manager.
Declared In	SyncCommon.h
Prototype	<code>typedef UInt32 CONDHANDLE</code>
Compatibility	Sync Manager version: All. Palm OS version: All.
See Also	<u>SyncRegisterConduit()</u> , <u>SyncUnRegisterConduit()</u>

DBDatabaseInfo Struct

Purpose Defines information in a database header. The defined fields depend on whether the database is a schema, extended, or classic database.

Declared In SyncCommon.h

Prototype

```
typedef struct DBDatabaseInfo {
    // Section 1:
    // Basic fields for all database types.
    UInt32 type;
    UInt32 creator;
    UInt16 attributes;
    UInt16 flags;
    UInt16 version;
    UInt32 createDate;
    UInt32 modifyDate;
    UInt32 backupDate;
    UInt32 modifyNumber;
    char name[dmDBNameLength];
    // Section 2:
    // Retrieved by SyncDbFindDatabase\(\) and
    // SyncDbFindDatabaseByTypeCreator\(\) functions
    // for all database types.
    UInt32 rowCount;
    UInt32 totalBytes;
    UInt32 dataBytes;
    // Section 3:
    // Retrieved by SyncDbReadOpenDatabaseInfo\(\)
    // for schema databases only.
    UInt16 encoding;
    UInt32 tableCount;
    char dispName[dmDBNameLength];
    // Section 4:
    // Retrieved by SyncDmReadOpenDatabaseInfo\(\) for
    // non-schema databases only.
    UInt32 appBlkSize;
    UInt32 sortBlkSize;
    UInt32 maxRecSize;
} DBDatabaseInfo
```

Fields type

Defines the four-byte Palm OS [database type](#).

Common Sync Manager API

DBDatabaseInfo

`creator`

Defines the four-character [creator ID](#) for the database.

`attributes`

Defines the attributes of the database. This field is a combination of one or more “[Database Attributes](#)” on page 407.

`flags`

Defines whether the database is excluded from HotSync operations and whether the database is in RAM on the handheld. This field is a combination of one or more of the values described in “[Database Information Flags](#)” on page 411.

`version`

Defines the application-specific version number. The developer defines this version number for the database, which Palm OS can use to determine whether a newer version of a database can overwrite an older one. The default version number is 0.

`createDate`

Receives the date on which the database is created. This is a `time_t` value. See the “Comments” section below.

`modifyDate`

Receives the date of the most recent database modification. This is a `time_t` value. See the “Comments” section below.
Note: versions 1.x of Palm OS did not update the modification date.

`backupDate`

Receives the date of the most recent database backup. This is a `time_t` value. See the “Comments” section below.

`modifyNumber`

Receives the database modification number, which is incremented every time a record in the database is added, modified, or deleted on the handheld.

`name`

Defines the null-terminated [database name](#) as a character array. Palm OS Cobalt uses the display name of a schema database (defined by the `dispname` field), if it is defined; otherwise, they use the internal name (defined by the `name` field).

rowCount

Receives the number of rows in a schema database.

totalBytes

Receives the total number of bytes of storage used by a schema database, including overhead. Contrast with the dataBytes field.

dataBytes

Receives the total number of bytes of storage used by a schema database for data only, excluding overhead. Contrast with the totalBytes field.

encoding

(*Schema databases only*) Defines the type of character encoding of text data in a schema database. This field is one of the values defined in “[Character Encoding Types](#)” on page 31.

tableCount

Defines the number of tables in a schema database.

dispName

(*Schema databases only*) Defines the null-terminated display **database name** as a character array. Palm OS Cobalt uses the display name of a database, if it is defined; otherwise, they use the internal name (defined by the name field).

appBlkSize

(*Non-schema databases only*) Receives the block size, in bytes, of the application info block in an extended database. This field is filled in only by [SyncDmReadOpenDatabaseInfo\(\)](#) when the SYNC_DB_INFO_OPT_GET_SIZE option is set.

sortBlkSize

(*Non-schema databases only*) Receives the block size, in bytes, of the sort info block in a classic database. This field is filled in only by [SyncDmReadOpenDatabaseInfo\(\)](#) when the SYNC_DB_INFO_OPT_GET_SIZE option is set.

maxRecSize

(*Non-schema databases only*) Receives the size of the largest record in an extended database. This field is filled in only by [SyncDmReadOpenDatabaseInfo\(\)](#) when the SYNC_DB_INFO_OPT_GET_MAX_REC_SIZE option is set.

Common Sync Manager API

DBDatabaseInfo

Comments	The createDate, modifyDate, and backupDate fields are time_t values in GMT for schema and extended databases; for classic databases, these are time_t values in the handheld's local time.
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncReadDatabaseList() , SyncDbFindDatabase() , SyncDbFindDatabaseByTypeCreator() , SyncDbReadOpenDatabaseInfo() , SyncDmFindDatabase() , SyncDmFindDatabaseByTypeCreator() , SyncDmReadOpenDatabaseInfo() , SyncIsDatabaseBackupNeeded() , SyncGenerateBackupFileName() , SyncReadBackupImageInfo() , SyncInstallDatabase() , SyncInstallAndBackupDatabase() ,

Common Sync Manager Constants

This section describes the following enumerated and preprocessor constants, which are used by either schema, extended, classic, or common Sync Manager functions.

Database Attributes	Define the set of attributes that a database can have. These attributes apply to schema, extended, and classic databases.
Database Closing Options	Indicate optional actions for <code>SyncDbCloseDatabase()</code> , <code>SyncDmCloseDatabaseEx()</code> , or <code>SyncCloseDBEx()</code> to take when it closes a database.
Database Information Flags	Define values for the <code>flags</code> field of the <code>DBDatabaseInfo</code> structure.
Database Information Retrieval Options	Indicate how Sync Manager's find functions retrieve data about a database.
Database Search Options	Indicate how Sync Manager "find" functions performs a search operation.
Deprecated Constants	Deprecated versions of newer constants. Use the newer constant that each is defined as below.
eConnType	Indicates whether the handheld is connected to the desktop computer via a local, fast connection or a remote, possibly slow connection.
eDbFlags	Defines the set of attributes that a database can have.
eDbOpenModes	Indicates what modes in which to open a database.
eDesktopTrustStatus	Indicates the desktop trust status of the HotSync operation that is in progress.
eFirstSync	Indicates to a conduit whether the handheld has previously been synchronized with the desktop computer.

Common Sync Manager API

Common Sync Manager Constants

<u>eMiscDbListFlags</u>	Indicates miscellaneous properties of a database in a CDbList object.
<u>eSyncPref</u>	Indicates whether the user preferences specified in a CSyncPreference object or CfgConduitInfoType structure apply temporarily or permanently.
<u>eSyncRecAttrs</u>	Indicates the record modification attributes that can be combined in the CRawRecordInfo m_Attribs member.
<u>eSyncTypes</u>	Indicates the type of HotSync operation to perform, either from HotSync Manager to the conduit or from the conduit's Custom dialog box to HotSync Manager.
<u>Maximum Buffer Sizes</u>	Define the maximum size of various buffer used by some Sync Manager functions.
<u>Miscellaneous Constants</u>	Define constants used with various Sync Manager functions.
<u>Record Attributes</u>	Define attributes of records in a non-schema database.
<u>Versions of the Sync Manager API</u>	Define the major and minor version numbers of the Sync Manager API returned by SyncGetAPIVersion().

Database Attributes

Purpose	Define the set of attributes that a database can have. These attributes apply to schema, extended, and classic databases.
Declared In	<code>SyncCommon.h</code>
Constants	<pre>#define dmAllHdrAttrs (dmHdrAttrResDB dmHdrAttrReadOnly dmHdrAttrAppInfoDirty dmHdrAttrBackup dmHdrAttrOKToInstallNewer dmHdrAttrResetAfterInstall dmHdrAttrCopyPrevention dmHdrAttrStream dmHdrAttrHidden dmHdrAttrLaunchableData dmHdrAttrRecyclable dmHdrAttrBundle dmHdrAttrSchema dmHdrAttrSecure dmHdrAttrFixedUp dmHdrAttrOpen)</pre> <p>A mask used to specify all header attributes.</p> <pre>#define dmHdrAttrAppInfoDirty 0x0004</pre> <p><i>(Nonschema databases only)</i> The application info block is dirty (it has been modified since the last HotSync operation).</p> <pre>#define dmHdrAttrBackup 0x0008</pre> <p>The database should be backed up to the desktop computer if no application-specific conduit is available.</p> <pre>#define dmHdrAttrBundle 0x0800</pre> <p>The database is bundled during a beam with the application of the same creator ID. 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.</p> <p>This attribute applies to Palm OS® versions 4.0 and later. Note that overlay databases are automatically beamed with the application database. You do not need to set this bit in overlay databases.</p> <pre>#define dmHdrAttrClassic 0x0000</pre> <p>The database is a classic record database. Only Palm OS can set this attribute.</p> <pre>#define dmHdrAttrCopyPrevention 0x0040</pre> <p>Prevents the database from being copied by methods such as IR beaming.</p>

Common Sync Manager API

Database Attributes

```
#define dmHdrAttrExtended (dmHdrAttrSecure)
    The database is an extended database. Note that this attribute
    has the same value as dmHdrAttrSecure. If the database is
    a not a schema database, this attribute determines whether it
    is an extended database or a classic database.

#define dmHdrAttrFixedUp 0x4000
    The Palm OS loader had to fix up an application for
    relocation. Only Palm OS can set this attribute.

#define dmHdrAttrHidden 0x0100
    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 non-resource databases to have
    the Launcher disregard the database's rows or records when
    when it shows a count of rows or records in its Info dialog.

#define dmHdrAttrLaunchableData 0x0200
    This database (not applicable for executables) can be
    "launched" from the Launcher, which passes the database's
    name to its owner application ('appl' database with same
    creator ID) using the sysAppLaunchCmdOpenNamedDB
    action code.

#define dmHdrAttrOKToInstallNewer 0x0010
    A backup conduit can install a newer version of this database
    with a different name if the current database is open.

#define dmHdrAttrOpen 0x8000
    The database is open. Only Palm OS can set this attribute.

#define dmHdrAttrReadOnly 0x0002
    The database is a read-only database.

#define dmHdrAttrRecyclable 0x0400
    The database is recyclable. Recyclable databases are deleted
    when they are closed or upon a system reset.

#define dmHdrAttrResDB 0x0001
    The database is a resource database. Only Palm OS can set
    this attribute.

#define dmHdrAttrResetAfterInstall 0x0020
    The handheld must be reset after this database is installed.
    That is, the HotSync application on the handheld forces a
    reset after installing this database.
```

```
#define dmHdrAttrSchema 0x1000
    The database is a schema database. Only Palm OS can set this
    attribute.

#define dmHdrAttrSecure 0x2000
    The database is a secure database. Note that this attribute has
    the same value as dmHdrAttrExtended. If the database is a
    schema database, this attribute determines whether it is a
    secure database. Only Palm OS can set this attribute.

#define dmHdrAttrStream 0x0080
    The database is a file stream.

#define dmSysOnlyHdrAttrs ( dmHdrAttrResDB |
    dmHdrAttrSchema | dmHdrAttrSecure |
    dmHdrAttrFixedUp | dmHdrAttrOpen )
    A mask specifying the attributes that only the system can
    change.
```

Comments The attributes field in a [DBDatabaseInfo](#) structure can have any valid combination of these values.
For a description of the different types of databases indicated by some of these attributes, see [Chapter 8, “Palm OS Databases,”](#) on page 113 in *Introduction to Conduit Development*.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [DBDatabaseInfo](#), [SyncReadDatabaseList\(\)](#),
[SyncDbFindDatabase\(\)](#),
[SyncDbFindDatabaseByTypeCreator\(\)](#),
[SyncDbReadOpenDatabaseInfo\(\)](#),
[SyncIsDatabaseBackupNeeded\(\)](#),
[SyncGenerateBackupFileName\(\)](#),
[SyncReadBackupImageInfo\(\)](#),
[SyncInstallAndBackupDatabase\(\)](#),
[SyncInstallDatabase\(\)](#)

Common Sync Manager API

Database Closing Options

Database Closing Options

Purpose	Indicate optional actions for SyncDbCloseDatabase() , SyncDmCloseDatabaseEx() , or SyncCloseDBEx() to take when it closes a database.
Declared In	<code>SyncCommon.h</code>
Constants	<pre>#define SYNC_CLOSE_DB_OPT_UPDATE_BACKUP_DATE 0x80 Indicates that the database's backup date is to be updated after it is closed. #define SYNC_CLOSE_DB_OPT_UPDATE_MOD_DATE 0x40 Indicates that the database's modification date is to be updated after it is closed.</pre>
Comments	For schema databases, an additional closing option is described in “ Database Modification Reset Flag ” on page 33 for use only with SyncDbCloseDatabase() .
Compatibility	Sync Manager version: 2.2 or later. Palm OS version: See SyncDbCloseDatabase() , SyncDmCloseDatabaseEx() , and SyncCloseDBEx() .
See Also	SyncDbCloseDatabase() , SyncDmCloseDatabaseEx() , SyncCloseDBEx()

Database Information Flags

Purpose	Define values for the <code>flags</code> field of the DBDatabaseInfo structure.
Declared In	<code>SyncCommon.h</code>
Constants	<pre>#define dbFlagExcludeFromSync ((UInt16)0x0080) Indicates that the database is to be excluded from the synchronization operations. This is typically the result of the user disabling synchronization for the application associated with the database on the handheld (accessible from the HotSync client's Options > Conduit Setup menu item).</pre> <pre>#define dbFlagRamBased ((UInt16)0x0040) Indicates that the database is located in RAM. If this flag is not set, the database is stored in ROM.</pre>
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	DBDatabaseInfo

Common Sync Manager API

Database Information Retrieval Options

Database Information Retrieval Options

Purpose	Indicate how Sync Manager's find functions retrieve data about a database.
Declared In	<code>SyncCommon.h</code>
Constants	<pre>#define SYNC_DB_INFO_OPT_GET_ATTRIBUTES (0x80)</pre> Indicates that database search operations are to retrieve database attribute information. <pre>#define SYNC_DB_INFO_OPT_GET_MAX_REC_SIZE (0x20)</pre> Indicates that database search operations are to retrieve the maximum record (or resource) size information. This option is available only to SyncReadOpenDbInfo() for classic databases and SyncDmReadOpenDatabaseInfo() for extended databases. <pre>#define SYNC_DB_INFO_OPT_GET_SIZE (0x40)</pre> Indicates that database search operations are to retrieve record (or resource) count and data size information.
Comments	As a performance optimization, omit any or all of these options so that find operations do not return more information than the caller needs. For schema databases, an additional retrieval option is defined in “Database Information Retrieval Option for SyncDbReadOpenDatabaseInfo()” on page 34.
Compatibility	Sync Manager version: 2.2 or later. Palm OS version: 3.0 or later.
See Also	SyncFindDbByName() , SyncFindDbByTypeCreator() , SyncReadOpenDbInfo() , SyncDatabaseInfoType , SyncFindDbByNameParams , SyncFindDbByTypeCreatorParams , SyncReadOpenDbInfoParams , SyncDbReadOpenDatabaseInfo()

Database Search Options

Purpose	Indicate how Sync Manager “find” functions performs a search operation.
Declared In	<code>SyncCommon.h</code>
Constants	<pre>#define SYNC_DB_SRCH_OPT_NEW_SEARCH (0x80) Indicates that a new search is to be started. In subsequent iterations with the same search criteria, the caller must exclude this flag. #define SYNC_DB_SRCH_OPT_ONLY_LATEST (0x40) Indicates that the search is for the latest version.</pre>
Compatibility	Sync Manager version: 2.2 or later. Palm OS version: 3.0 or later.
See Also	SyncDmFindDatabaseByTypeCreator() , SyncFindDbByTypeCreator() , SyncFindDbByTypeCreatorParams

Common Sync Manager API

Deprecated Constants

Deprecated Constants

Purpose	Deprecated versions of newer constants. Use the newer constant that each is defined as below.
Declared In	<code>SyncCommon.h</code>
Constants	<pre>#define DB_NAMELEN SYNC_DB_NAMELEN Use defined value in "Maximum Buffer Sizes" on page 429. #define eExcludeFromSync eMiscDbFlagExcludeFromSync Defined for backward compatibility. Use the newer name defined in "eMiscDbListFlags" on page 423. #define eModem eModemConnType Defined for backward compatibility. Use the newer name defined in "eConnType" on page 415. #define PASSWORD_LENGTH SYNC_REMOTE_PASSWORD_BUF_SIZE Use defined value in "Maximum Buffer Sizes" on page 429. #define REMOTE_CARDNAMELEN SYNC_REMOTE_CARDNAME_BUF_SIZE Use defined value in "Maximum Buffer Sizes" on page 429. #define REMOTE_MANUFNAMELEN SYNC_REMOTE_MANUFNAME_BUF_SIZE Use defined value in "Maximum Buffer Sizes" on page 429. #define REMOTE_USERNAME SYNC_REMOTE_USERNAME_BUF_SIZE Use defined value in "Maximum Buffer Sizes" on page 429.</pre>
Compatibility	Sync Manager version: All. Palm OS version: All.

eConnType Enum

Purpose Indicates whether the handheld is connected to the desktop computer via a local, fast connection or a remote, possibly slow connection.

Declared In SyncCommon.h

Constants

eCable

Indicates that the handheld is connected locally with a fast connection, either with a direct USB or serial cable or through a local area network.

eModemConnType

Indicates that the handheld is connected remotely, possibly with a slow connection—for example, via a modem.

eConnTypeDoNotUse = 0xFFFF

An invalid value used to make all compilers see the enum as a 4-byte value. Do not use.

Compatibility Sync Manager version: All.

Palm OS version: All.

See Also [CSyncProperties](#)

Common Sync Manager API

eDbFlags

eDbFlags Enum

Purpose	Defines the set of attributes that a database can have.
Declared In	SyncCommon.h
Constants	<code>eRecord = 0x0000</code> Indicates that the database is a record database. This is the default value and is mutually exclusive with <code>eResource</code> . <code>eResource = 0x0001</code> Indicates that the database is a resource database. If this flag is not specified, the database is considered a record database. Most conduits access only record databases. <code>eReadOnly = 0x0002</code> Indicates that the database is a read-only database in ROM. <code>eAppInfoDirty = 0x0004</code> Indicates that the database's application info block has been modified. Not valid for schema databases. <code>eBackupDB = 0x0008</code> Indicates that the database is to be backed up to the desktop computer if no application-specific conduit is registered with the same creator ID. <code>eOkToInstallNewer = 0x0010</code> Indicates that the Backup/Restore conduit can install a newer version of the database with a different name if the current database is currently opened. <code>eResetAfterInstall = 0x0020</code> Indicates that the handheld needs to be reset after the HotSync operation during which the database is installed. (actually, after the synchronization involving the database is complete). <code>eCopyPrevention = 0x0040</code> Indicates that the database is not to be copied or beamed to other handhelds. Copy prevention is supported in Palm OS versions 3.0 and later. <code>eStream = 0x0080</code> Indicates that the database is used for file stream implementation.

eHidden = 0x0100

Indicates that the database should generally be hidden from view. This flag is used to hide some applications from the main view of the Launcher, for example. For record (not resource) databases, this flag hides the record count within the Launcher's Info form.

eLaunchableData = 0x0200

Indicates that the record database (not applicable for executables) can be "launched" by passing its name to its owner application ('app1' database with same creator ID) using the sysAppLaunchCmdOpenNamedDB action code.

eRecyclable = 0x0400

Indicates that the database (resource or record) is "recyclable"—that is, it will be deleted very soon, generally the next time the database is closed.

eBundle = 0x0800

Indicates that the database (resource or record) is associated with the application with the same creator ID. It will be beamed and copied along with the application.

This attribute applies to Palm OS® versions 4.0 and later. Note that overlay databases are automatically beamed with the application database. You do not need to set this bit in overlay databases.

eSchema = 0x1000

Indicates that the database is a schema database, not a classic or extended database. Schema databases are supported only in Palm OS Cobalt.

eSecure = 0x2000

Indicates that the database is a secure database if eSchema is also set; otherwise it indicates an extended database. Secure databases and extended databases are supported only in Palm OS Cobalt.

eOpenDB = 0x8000

Indicates that the database is currently opened.

Note: Do not pass this flag when creating a database. It is for system use only!

Common Sync Manager API

eDbFlags

eDbFlagsDoNotUse = 0xFFFF

An invalid value used to make all compilers see the enum as a 4-byte value. Do not use.

Comments	<p>The caller can combine these flags together to specify information about a database. Other functions pass back a combination of these values to indicate the properties of a database. Note that the eRecord and eResource flags are mutually exclusive and that you must specify exactly one of them when creating a database.</p> <p>For a schema database, eSchema and eRecord must be set and eResource must not be set; if eSecure is also set, then it is a secure schema database.</p> <p>For a non-schema database, eSchema is not set. In this case, eSecure indicates which type of non-schema database it is: if set, an extended database; if not set, a classic database. For non-schema databases, either eRecord or eResource can be set.</p> <p>For a summary of mutually exclusive characteristics, see “Mutually Exclusive Database Characteristics” on page 116 in the <i>Introduction to Conduit Development</i>.</p>
Compatibility	Sync Manager version: All. Palm OS version: See each constant definition.
See Also	CDBCreateDB , CDBList , SyncReadOpenDbInfoParams , SyncCreateDB() , SyncReadOpenDbInfo() , SyncDmCreateDatabase() , SyncFindDbByName() , SyncFindDbByTypeCreator()

eDbOpenModes Enum

Purpose	Indicates what modes in which to open a non-schema database.
Declared In	SyncCommon.h
Constants	<p><code>eDbShowSecret = 0x0010</code> Indicates to open the database with full access to the user's secret records. <i>Note:</i> only two Sync Manager functions are currently affected by this mode: the SyncReadNextRecInCategory() and SyncReadNextModifiedRecInCategory() functions skip secret records if you open the database without specifying the <code>eDbShowSecret</code> flag.</p> <p><code>eDbExclusive = 0x0020</code> Indicates to open the database with exclusive access for the caller. This means that if the database is already opened in this mode, the caller is denied access. If the caller successfully opens the database in exclusive mode, no other caller can access it until the exclusive caller closes the database.</p> <p><code>eDbWrite = 0x0040</code> Indicates to open the database for write access.</p> <p><code>eDbRead = 0x0080</code> Indicates to open the database for read access.</p> <p><code>eDbOpenModesDoNotUse = 0xFFFF</code> An invalid value used to make all compilers see the enum as a 4-byte value. Do not use.</p>
Comments	The following rules explain how to use the database open mode constants: <ul style="list-style-type: none">• Generally include the <code>eDbShowSecret</code> flag; otherwise, some of the Sync Manager functions do not return records that are marked as private.• To open a database for reading only, specify (<code>eDbRead eDbShowSecret</code>).• To open a database for reading and writing, specify (<code>eDbRead eDbWrite eDbShowSecret</code>).• Use the <code>eDbExclusive</code> flag to open the database in exclusive mode, which means that nothing else on the handheld can use the database until the exclusive caller

Common Sync Manager API

eDbOpenModes

closes it. This also means that a call to open fails if anything else on the handheld is using the database.

Note that this enum is used only with functions that open non-schema databases. See “[Database Open Modes](#)” on page 37 for modes in which to open schema databases.

Compatibility Sync Manager version: All.
Palm OS version: All.

See Also [SyncDmOpenDatabase \(\)](#), [SyncOpenDB \(\)](#)

eDesktopTrustStatus Enum

Purpose	Indicates the desktop trust status of the HotSync operation that is in progress.
Declared In	SyncCommon.h
Constants	<p>eDesktopNotTrusted Indicates that the user does not allow this desktop to open secure databases.</p> <p>eDesktopTrusted Indicates that the user allows this desktop to open secure databases.</p> <p>eDesktopTrustNotVerified Indicates that the Sync Manager cannot verify whether the user allows this desktop to open secure databases.</p>
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	SyncGetDesktopTrustStatus()

Common Sync Manager API

eFirstSync

eFirstSync Enum

Purpose	Indicates to a conduit whether the handheld has previously been synchronized with the current Windows user on this desktop computer.
Declared In	<code>SyncCommon.h</code>
Constants	<p><code>eNeither</code> Indicates that the handheld has been synchronized before with this Windows user: the handheld's HotSync user ID matches one of HotSync user IDs for the current Windows user on this desktop computer.</p> <p><code>ePC</code> Indicates that the handheld has been synchronized before, but has never been synchronized by the current Windows user on this desktop computer: the handheld has a HotSync user ID that does not match any HotSync user ID for the current Windows user on this desktop computer.</p> <p><code>eHH</code> Indicates that handheld has not been synchronized before and therefore does not have a HotSync user ID. This might indicate a handheld that was recently reset or synchronized with a user profile.</p> <p><code>eFirstSyncDoNotUse = 0xFFFF</code> An invalid value used to make all compilers see the enum as a 4-byte value. Do not use.</p>
Comments	<p>HotSync Manager passes one of these values to a conduit when it calls the conduit's OpenConduit() entry point. Conduits use this value along with <code>CSyncProperties::m_SyncType</code> (and with SyncDbGetSyncMode() for schema databases) to determine the actual sync mode.</p> <p>Note that HotSync Manager versions earlier than 6.0 share the same set of HotSync users across all Windows users on a given desktop computer. So in these earlier versions, <code>eNeither</code> and <code>ePC</code> indicate that the handheld has been synchronized with the current desktop computer regardless of the current Windows user.</p>
Compatibility	Sync Manager version: All. Palm OS version: All.
See Also	CSyncProperties , OpenConduit() , SyncDbGetSyncMode()

eMiscDbListFlags Enum

Purpose	Indicates miscellaneous properties of a database in a CDBList object.
Declared In	SyncCommon.h
Constants	<p><code>eMiscDbFlagExcludeFromSync = 0x0080</code> Indicates that the database is to be excluded from the synchronization operations. This is typically the result of the user disabling synchronization for the application associated with the database on the handheld (accessible from the HotSync client's Options > Conduit Setup menu item). This feature is supported in Palm OS versions 2.0 or later. This constant replaces the older constant <code>eExcludeFromSync</code>.</p> <p><code>eMiscDbFlagRamBased = 0x0040</code> Indicates that the database is located in RAM. If this flag is not set, the database is stored in ROM. This flag is available with Palm OS versions 3.0 or later.</p> <p><code>eMiscDbFlagsDoNotUse = 0xFFFF</code> An invalid value used to make all compilers see the enum as a 4-byte value. Do not use.</p>
Comments	These enum values are used in the <code>m_miscFlags</code> field of the CDBList class. Note that the <code>flags</code> field in the DBDatabaseInfo structure uses the constants defined in " Database Information Flags " on page 411, which are essentially the same as these enum values.
Compatibility	Sync Manager version: All. Palm OS version: All.
See Also	CDBList

Common Sync Manager API

eSyncPref

eSyncPref Enum

Purpose	Indicates whether the user preferences specified in a CSyncPreference object or CfgConduitInfoType structure apply temporarily or permanently.
Declared In	<code>SyncCommon.h</code>
Constants	<p><code>eNoPreference</code> Indicates that no user preferences are specified.</p> <p><code>ePermanentPreference</code> Indicates that the preference is permanent and therefore applies to all future HotSync operations.</p> <p><code>eTemporaryPreference</code> Indicates that the preference is temporary and only applies to the next HotSync operation.</p> <p><code>eSyncPrefDoNotUse = 0xFFFF</code> An invalid value used to make all compilers see the enum as a 4-byte value. Do not use.</p>
Compatibility	Sync Manager version: All. Palm OS version: All.
See Also	CSyncPreference , CfgConduitInfoType

eSyncRecAttrs Enum

Purpose	Indicates the record modification attributes that can be combined in the CRawRecordInfo m_Attrbs member.
Declared In	SyncCommon.h
Constants	<p>eRecAttrDeleted = 0x80 Indicates that the record has been marked as deleted on the handheld. This replaces the older constant DELETE_BIT.</p> <p>eRecAttrDirty = 0x40 Indicates that the record has been marked as modified. This replaces the older constant DIRTY_BIT.</p> <p>eRecAttrBusy = 0x20 Indicates that the record is in use by an application on the handheld. <i>Note:</i> This attribute is for system use only. Conduits must treat it as read-only and must <i>not</i> set it when writing records.</p> <p>eRecAttrSecret = 0x10 Indicates that the record has been marked as private and should be displayed only if the user wishes. This replaces the older constant PRIVATE_BIT.</p> <p>eRecAttrArchived = 0x08 Indicates that the record has been marked for archiving. This replaces the older constant ARCHIVE_BIT.</p> <p>eSyncRecAttrDoNotUse = 0xFFFF An invalid value used to make all compilers see the enum as a 4-byte value. Do not use.</p>
Comments	These values are essentially the same as those defined in “ Record Attributes ” on page 432.
Compatibility	Sync Manager version: All. Palm OS version: All.
See Also	CRawRecordInfo , SyncReadRecordById() , SyncReadRecordByIndex() , SyncReadNextModifiedRec() , SyncReadNextModifiedRecInCategory() , SyncReadNextRecInCategory() , SyncResetSyncFlags() , SyncWriteRec()

Common Sync Manager API

eSyncTypes

eSyncTypes Enum

Purpose	Indicates the type of HotSync operation to perform, either from HotSync Manager to the conduit or from the conduit's Custom dialog box to HotSync Manager.
Declared In	<code>SyncCommon.h</code>
Constants	<p><code>eFast</code> Indicates a fast synchronization: only records that have been added, archived, deleted, or modified are exchanged between the handheld and the desktop computer. Note: this value indicates only that this handheld's last HotSync operation was with the current desktop. This is sufficient for determining how to synchronize non-schema databases, but it is insufficient for schema databases. See "Comments" below.</p> <p><code>eSlow</code> Indicates a slow synchronization: every record is read from the handheld and compared to the corresponding record on the desktop computer (in the desktop data file and its backup file). HotSync Manager indicates this type of synchronization when the handheld has been synchronized a different desktop. See "Comments" below.</p> <p><code>eHHToPC</code> Indicates a restore from the handheld: overwrite the desktop database with the database from the handheld.</p> <p><code>ePCToHH</code> Indicates a restore from the desktop computer: overwrite the database on the handheld with the database on the desktop computer.</p> <p><code>eInstall</code> Indicates that HotSync Manager is calling the conduit's OpenConduit() entry point during an install phase of a HotSync operation. This is the synchronization type for install conduits.</p> <p><code>eBackup</code> Indicates that HotSync Manager is calling the backup conduit's OpenConduit() entry point during the backup phase of a HotSync operation. This is the synchronization type for a backup conduit.</p>

eDoNothing

Indicates that the conduit does not exchange data between the handheld and the desktop computer; however, the conduit is loaded and can set flags or log messages.

eProfileInstall

Indicates a user profile download. A [user profile](#) is a special user account that you can set up on the desktop computer that downloads data to a handheld, without assigning a user ID.

eSyncTypeDoNotUse = 0xFFFF

An invalid value used to make all compilers see the enum as a 4-byte value. Do not use.

Comments

These values are used in the following ways:

- When HotSync Manager calls a conduit's [OpenConduit\(\)](#) entry point, it passes the conduit one of these values in a [CSyncProperties](#) object to indicate the type of the current HotSync operation.

The Sync Manager passes back the eFast or esSlow value based solely on whether this handheld's last HotSync operation was with the current desktop. For non-schema databases, this is sufficient for a conduit to determine whether to perform a fast or slow sync. However, for schema databases, this value is not sufficient. Instead of relying on this value to determine whether to perform a fast or slow sync, conduits that synchronize schema databases must call [SyncDbGetSyncMode\(\)](#). Conduits synchronizing schema databases must still rely on this field for the other values (eHHToPC, ePCToHH, etc.) it receives, which are equally valid for all database types.

- When HotSync Manager calls a conduit's [ConfigureConduit\(\)](#) entry point, it passes the conduit one of these values in a [CSyncPreference](#) object to indicate the user's stored preference for the next HotSync operation.
- When HotSync Manager calls a conduit's [CfgConduit\(\)](#) entry point, it passes the conduit one of these values in each of three fields of a [CfgConduitInfoType](#) structure to indicate the user's stored preference for the next HotSync operation.

Common Sync Manager API

eSyncTypes

- When HotSync Manager calls a conduit's [GetConduitInfo\(\)](#) entry point and passes in *infoType* = *eDefaultAction*, the conduit passes back one of these values to indicate the default HotSync operation to perform.

Compatibility Sync Manager version: 2.0 or later.
Palm OS version: All.

See Also [CSyncProperties](#), [CSyncPreference](#), [CfgConduitInfoType](#), [GetConduitInfo\(\)](#)

Maximum Buffer Sizes

Purpose Define the maximum size of various buffer used by some Sync Manager functions.

Declared In SyncCommon.h

Constants

```
#define BIG_PATH 256
      The maximum size in bytes of various buffers that hold paths
      and other names.

#define SYNC_DB_NAMELEN (32)
      The maximum size of a handheld database name, including
      the null terminator character. This constant replaces the older
      constant DB_NAMELEN.

#define SYNC_MAX_HH_LOG_SIZE (2*1024)
      The maximum size in bytes of the HotSync log on the
      handheld.

#define SYNC_MAX_PROD_ID_SIZE (255)
      The maximum number of bytes in the product ID buffer.

#define SYNC_MAX_USERNAME_LENGTH (20)
      The maximum length in bytes of a user name on the
      handheld (not including the null terminator character).

#define SYNC_REMOTE_CARDNAME_BUF_SIZE (32)
      The buffer size in bytes for the name of the memory card on
      the handheld. This constant replaces the older constant
      REMOTE_CARDNAMELEN.

#define SYNC_REMOTE_MANUFNAME_BUF_SIZE (32)
      The buffer size in bytes for the manufacturer name of the
      memory card on the handheld. This constant replaces the
      older constant REMOTE_MANUFNAMELEN.

#define SYNC_REMOTE_PASSWORD_BUF_SIZE (64)
      The buffer size in bytes for the password on the handheld.
      This constant replaces the older constant
      PASSWORD_LENGTH.

#define SYNC_REMOTE_USERNAME_BUF_SIZE (64)
      The buffer size in bytes for the user name on the handheld.
      This constant replaces the older constant
      REMOTE_USERNAME.
```

Common Sync Manager API

Maximum Buffer Sizes

Compatibility Sync Manager version: 2.0 or later.
Palm OS version: All.

Miscellaneous Constants

Purpose	Define constants used with various Sync Manager functions.
Declared In	<code>SyncCommon.h</code>
Constants	<pre>#define dmDBNameLength (32)</pre> <p>The maximum number of ASCII bytes in a database name, including NULL terminator.</p> <pre>#define kOffsetEndOfData (0xFFFFFFFF)</pre> <p>Specifies that a write row operation—like SyncDmWriteRecord() or SyncDbWriteColumnValue()—appends new data to the end of existing. Pass this constant in the <i>dataOffset</i> parameter. Using this constant is equivalent to specifying an offset equal to the data size, but does not require a previous call to retrieve the size when the size is not known in advance.</p> <pre>#define SIZEOF_DB_DATABASE_INFO (sizeof(DBDatabaseInfo))</pre> <p>Defines the size of the DBDatabaseInfo structure.</p>
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.

Common Sync Manager API

Record Attributes

Record Attributes

Purpose	Define attributes of records in a non-schema database.
Declared In	SyncCommon.h
Constants	<pre>#define dmAllRecAttrs (dmRecAttrDelete dmRecAttrDirty dmRecAttrBusy dmRecAttrSecret)</pre> <p>A combination of flags that indicates all record attributes. This value is useful when checking an attribute value for errors.</p> <pre>#define dmRecAttrBusy 0x20</pre> <p>Indicates that the record is in use by an application on the handheld. <i>Note:</i> This attribute is for system use only. Conduits must treat it as read-only and must <i>not</i> set it when writing records.</p> <pre>#define dmRecAttrDelete 0x80</pre> <p>Indicates that the record has been marked deleted.</p> <pre>#define dmRecAttrDirty 0x40</pre> <p>Indicates that the record has been marked as modified.</p> <pre>#define dmRecAttrSecret 0x10</pre> <p>Indicates that the record has been marked as private and should be displayed only if the user wishes.</p> <pre>#define dmSysOnlyRecAttrs (dmRecAttrBusy)</pre> <p>Mask that identifies those attributes that only the system can change.</p>
Comments	These values are used in CRawRecordInfo.m_Attrib and passed back in the <i>pAttributes</i> parameter of extended Sync Manager read functions—for example, SyncDmReadRecordByID() .
Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.

Versions of the Sync Manager API

Purpose	Define the major and minor version numbers of the Sync Manager API returned by SyncGetAPIVersion() .
Declared In	SyncCommon.h
Constants	<pre>#define SYNCAPI_VER_MAJOR_2 2 Indicates the major version number is 2. #define SYNCAPI_VER_MINOR_0 0 Indicates the minor version number is 0. #define SYNCAPI_VER_MINOR_1 1 Indicates the minor version number is 1. #define SYNCAPI_VER_MINOR_2 2 Indicates the minor version number is 2. #define SYNCAPI_VER_MINOR_3 3 Indicates the minor version number is 3. #define SYNCAPI_VER_MINOR_4 4 Indicates the minor version number is 4. #define SYNCAPI_VER_MINOR_5 5 Indicates the minor version number is 5.</pre>
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.
See Also	SyncGetAPIVersion() , Chapter 1, “Overview of the Sync Manager API,” on page 3

Common Sync Manager API

Common Sync Manager Functions and Macros

Common Sync Manager Functions and Macros

This section describes the following functions and macros, which work with more than one type of database (schema, extended, or classic) or that do not work with databases at all.

<u>SyncAddLogEntry()</u>	Adds an entry to the HotSync log on the handheld.
<u>SyncBackupDatabase()</u>	Backs up a handheld database to a directory or file on the desktop.
<u>SyncBackupSecurityData()</u>	Backs up security vault databases from the handheld to a directory on the desktop.
<u>SyncCallDeviceApplication()</u>	Calls an application on a Palm OS Cobalt handheld and returns data and status information to your conduit from that application.
<u>SyncGenerateBackupFileName()</u>	Generates the unique backup filename of a database specified by its name, creator, type, and attributes.
<u>SyncGetAPIVersion()</u>	Retrieves the version of the Sync Manager API that is installed on the desktop computer.
<u>SyncGetDesktopTrustStatus()</u>	Determines whether the HotSync operation in progress is with a trusted desktop.
<u>SyncGetHHOSVersion()</u>	Retrieves the version number of the operating system on the handheld.
<u>SyncHHToHostDWord()</u>	Returns the desktop computer's representation of the specified 32-bit UInt32 value from the handheld.
<u>SyncHHToHostWord()</u>	Returns the desktop computer's representation of the specified 16-bit UInt16 value from the handheld.

<u>SyncHostToHHDWord()</u>	Returns the handheld's representation of the specified 32-bit UInt32 value from the desktop computer.
<u>SyncHostToHHWord()</u>	Returns the handheld's representation of the specified 16-bit UInt16 value from the desktop computer.
<u>SyncInstallAndBackupDatabase()</u>	Installs a database on the handheld from an image file on the desktop and then backs up the same database.
<u>SyncInstallDatabase()</u>	Installs a database on the handheld from an image file on the desktop.
<u>SyncIsDatabaseBackupNeeded()</u>	Determines whether the desktop backup file for a database on the handheld is out-of-date.
<u>SyncLoopBackTest()</u>	(Private) Tests communications with the device.
<u>SyncReadBackupImageInfo()</u>	Reads the database header information from a backup image file on the desktop.
<u>SyncReadFeature()</u>	Retrieves a 32-bit feature value from the Feature Manager on the handheld.
<u>SyncReadSingleCardInfo()</u>	Retrieves information about a memory card on the handheld.
<u>SyncReadSysDateTime()</u>	Retrieves the current date and time, according to the system clock on the handheld.
<u>SyncReadSystemInfo()</u>	Retrieves the Palm OS software version and product information for the handheld.
<u>SyncReadUserID()</u>	Retrieves information about the user of the handheld, including the user name, last synchronization date, and encrypted password.
<u>SyncRebootSystem()</u>	Sends a request to soft-reset the handheld at the end of the HotSync operation.

Common Sync Manager API

Common Sync Manager Functions and Macros

[SyncRegisterConduit\(\)](#)

Registers a conduit that is about to start synchronization operations and returns a handle for use with other Sync Manager functions.

[SyncRestoreSecurityData\(\)](#)

Restores security vault databases from the desktop to the handheld.

[SYNCROMVMAJOR\(\)](#)

Decodes the major version number of Palm OS on the handheld.

[SYNCROMVMINOR\(\)](#)

Decodes the minor version number of Palm OS on the handheld.

[SyncUnRegisterConduit\(\)](#)

Unregisters a conduit that was successfully registered with a previous call to `SyncRegisterConduit()`, and cleans up any system resources that the Sync Manager allocated for the conduit.

[SyncWriteSysDateTime\(\)](#)

Sets the system date and time on the handheld.

[SyncYieldCycles\(\)](#)

Causes HotSync Manager to update its progress display indicator.

SyncAddLogEntry Function

Purpose	Adds an entry to the HotSync log on the handheld.
Declared In	SyncCommon.h
Prototype	SInt32 SyncAddLogEntry (const char * <i>pText</i>)
Parameters	<i>pText</i> The log string to add as a null-terminated character array.
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_MEM SYNCERR_REMOTE_BAD_ARG SYNCERR_LIMIT_EXCEEDED See " Common Sync Manager Error Codes " on page 486 for a description of all Sync Manager error codes.
Comments	Because the HotSync log on the handheld has limited space, keep your entries as short as possible. To include a new line in your log entry, use a single line-feed character (character code 0xA). Note that HotSync Manager automatically logs the general success or failure status of your conduit; therefore you need not add an entry for this purpose.
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.
See Also	LogAddEntry()

Common Sync Manager API

SyncBackupDatabase

SyncBackupDatabase Function

Purpose	Backs up a handheld database to a directory or file on the desktop.
Declared In	SyncCommon.h
Prototype	<pre>HSError SyncBackupDatabase (DBDatabaseInfo &dbInfo, TCHAR *filePath, HSBool bAlwaysBackup)</pre>
Parameters	<p>↔ <i>dbInfo</i> A reference to a DBDatabaseInfo structure that specifies values for the name, creator, type, and attributes of the database to back up. The type is only used as a cross-check and may be set to zero if you don't care what its value is. The attributes must specify whether the database is a classic, extended, or schema database. All other attributes bits are ignored. Upon successful return, receives values for all fields in section 1 of a DBDatabaseInfo structure for the matching database on the handheld, if it exists. Note that this function uses the type and attributes that it passes back to generate the backup filename, if <i>filePath</i> is a directory.</p> <p>↔ <i>filePath</i> The destination path or filename of the backup file, as a null-terminated TCHAR array. If the caller specifies a directory path, then the Sync Manager generates the filename automatically, appends it to the specified directory path, and passes it back upon return; in this case <i>filePath</i> must be large enough to contain the entire path (directory + filename). Do not pass in NULL.</p> <p>→ <i>bAlwaysBackup</i> If TRUE, always back up the database. If FALSE, backup the database only if necessary; for a description of how Sync Manager determines whether a database needs to be backed up, see SyncIsDatabaseBackupNeeded().</p>

Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following negative error code values:
	SYNCERR_ACCESS_DENIED Access to the specified database was denied by the handheld Authorization Manager—for example, because the database is secure and the desktop is not trusted.
	SYNCERR_ARG_MISSING
	SYNCERR_BAD_ARG One or more of the parameters passed in are invalid.
	SYNCERR_COMM_NOT_INIT
	SYNCERR_DISK_FULL The disk drive specified in <i>filePath</i> does not have enough free space.
	SYNCERR_FILE_ALREADY_OPEN
	SYNCERR_FILE_NOT_FOUND
	SYNCERR_LOCAL_CANCEL_SYNC
	SYNCERR_LOCAL_MEM This function was unable to allocate desktop memory.
	SYNCERR_LOST_CONNECTION
	SYNCERR_NOT_FOUND The specified database is not present on the handheld.
	SYNCERR_OPERATION_ABORTED
	SYNCERR_PATH_NOT_FOUND Neither <i>filePath</i> nor its parent directory exist.
	SYNCERR_REMOTE_BAD_ARG
	SYNCERR_REMOTE_MEM
	SYNCERR_REMOTE_NO_SPACE
	SYNCERR_REMOTE_SYS
	SYNCERR_TOO_MANY_OPEN_FILES Insufficient file handles are available on the desktop.

Common Sync Manager API

SyncBackupDatabase

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments

Based on the handheld database that you specify, this function creates or updates an image file of it on the desktop. This function enables a conduit to transfer an entire database to the desktop in a single call.

The caller specifies the database by its name, creator ID, type, and attributes (whether it is a classic, extended, or schema database).

This function works only with handhelds running Palm OS Cobalt.

If *filePath* is a directory, this function automatically generates the backup filename from the database creator, type, name, and attributes so that you do not have to call

[*SyncGenerateBackupFileName\(\)*](#) first.

This function backs up the database only if *all* of the following statements are true:

- The file or path (*filePath*) is writable.
- If the database is secure, the desktop must be trusted.
- *Either* of the following is true:
 - The corresponding call to [*SyncIsDatabaseBackupNeeded\(\)*](#) returns TRUE. This function tests backup dates, backup bit, whether the database and file exist and whether their names, creator IDs, and types match.
 - The *bAlwaysBackup* parameter is TRUE.

NOTE: If the database’s backup bit is not set, this function does not back it up, unless you set *bAlwaysBackup* to TRUE. But note that doing so causes this function to back up the database even if the specified image file on the desktop is already up to date.

Compatibility Sync Manager version: 2.5 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncIsDatabaseBackupNeeded\(\)](#),
[SyncGenerateBackupFileName\(\)](#),
[SyncBackupSecurityData\(\)](#), [SyncInstallDatabase\(\)](#)

Common Sync Manager API

SyncBackupSecurityData

SyncBackupSecurityData Function

Purpose	Backs up security vault databases from the handheld to a directory on the desktop.
Declared In	SyncCommon.h
Prototype	HSError SyncBackupSecurityData (TCHAR * <i>filePath</i>)
Parameters	<i>→ filePath</i> The destination directory of the vault files, as a null-terminated TCHAR array. The Sync Manager generates the filenames automatically. Do not pass in NULL.
Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following negative error code values: SYNCERR_ACCESS_DENIED Access is denied by the Authorization Manager on the handheld—for example, because it cannot access a secure database from a nontrusted desktop. SYNCERR_ARG_MISSING SYNCERR_BAD_ARG SYNCERR_COMM_NOT_INIT SYNCERR_DISK_FULL SYNCERR_FILE_ALREADY_OPEN SYNCERR_FILE_NOT_FOUND SYNCERR_LOCAL_CANCEL_SYNC SYNCERR_LOCAL_MEM This function was unable to allocate desktop memory. SYNCERR_LOST_CONNECTION SYNCERR_NOT_FOUND SYNCERR_OPERATION_ABORTED SYNCERR_PATH_NOT_FOUND The directory location pointed to by <i>filePath</i> does not exist. SYNCERR_REMOTE_BAD_ARG

SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_NO_SPACE
SYNCERR_REMOTE_SYS
SYNCERR_TOO_MANY_OPEN_FILES
SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments The Authorization Manager on the handheld stores all the relevant information to support secure databases—such as HEKs, rules, tokens, and so on—in one or more special secure databases called **vaults**. A backup conduit must back up vaults at the end of every HotSync session *after* all other databases. This function ensures that vaults are backed up in the order mandated by the Authorization Manager.

After a handheld is reset, vaults must be restored to the handheld *before* all other databases so that the Authorization Manager allows other secure databases to be restored afterwards. See [SyncRestoreSecurityData\(\)](#).

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also [SyncRestoreSecurityData\(\)](#)

Common Sync Manager API

SyncCallDeviceApplication

SyncCallDeviceApplication Function

Purpose	Calls an application on a Palm OS Cobalt handheld and returns data and status information to your conduit from that application.
Declared In	SyncCommon.h
Prototype	SInt32 SyncCallDeviceApplication (CCallApplicationParams *pParams)
Parameters	<i>↔ pParams</i> An object of the CCallApplicationParams class, which specifies information for the called application and receives information from the called application. On input, you must specify the application's creator ID, database name, and database attributes (classic, extended, or schema), as well as the action code and parameters to pass to the application and a result buffer and size used to received the return value from the application. Upon return, this object receives the result code and result size.
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_MEM SYNCERR_UNKNOWN_REQUEST The application on the handheld was not found, the application on did not handle the action code, or the handheld is running a version of Palm OS that this function does not support (see "Compatibility" below). SYNCERR_LOCAL_BUFF_TOO_SMALL The results buffer was not large enough to contain the results data. If this is the case, then upon return the value of <i>m_dwActResultSize</i> is greater than the value of <i>m_dwResultBufSize</i> , and only <i>m_dwResultBufSize</i> bytes were copied to the results buffer. See " Common Sync Manager Error Codes " on page 486 for a description of all Sync Manager error codes.

Comments	This function works only with handhelds running Palm OS Cobalt, version 6.0.1 or later. For these handhelds, use this function rather than SyncCallRemoteModule() . SyncCallDeviceApplication() allows you to uniquely identify the target application by creator ID, database name, and database attributes (classic, extended, or schema). Most conduits can accomplish their jobs without using this function. PalmSource recommends not using this function unless absolutely essential. An example of an unavoidable use of this function is when you need to call your application to create a secure database. For details, see “ Creating Secure Databases ” on page 134 in <i>Introduction to Conduit Development</i> .
	This function enables the caller to send arbitrary data in the <i>pParams</i> parameter to an application on the handheld during a HotSync operation. The application can pass back variable-sized information in <i>pParams</i> , which the caller can examine when this function returns. Note that the format of the data and the action codes are completely application-specific. The handheld application that you call must have the same structure as a Palm OS application; however, the application can have a proprietary type ID so that it does not show up in the Launcher. When a Palm OS Protein application is the target, you can set <i>pParams->m_wAttributes</i> to (dmHdrAttrSchema dmHdrAttrExtended) to indicate that the application may resided in either an extended or schema database. When a 68K application is the target, set <i>pParams->m_wAttributes</i> to zero. When <i>pParams->m_dwTypeID</i> is zero, SyncCallRemoteModule() launches the first application returned by DmFindDatabaseByTypeCreator(dmFindClassicDB) with type sysFileTApplication; otherwise if there is no such application, it launches the first application returned by DmFindDatabaseByTypeCreator(dmFindClassicDB) with type sysFileTPanel; otherwise it returns an error. So SyncCallRemoteModule() actually treats setting <i>m_dwTypeID</i> to zero as a substitute for first calling with a type ID of

Common Sync Manager API

SyncCallDeviceApplication

sysFileTApplication then a type ID of sysFileTPanel. SyncCallDeviceApplication() does not do this. Instead it simply passes m_dwTypeID to DmFindDatabaseByTypeCreator(), along with the findType determined by m_wAttributes. So if m_dwTypeID is zero, the wild card semantics of DmFindDatabaseByTypeCreator() apply. The SyncCallRemoteModule() semantics can be simulated by calling SyncCallDeviceApplication() twice with the appropriate types.

When you call SyncCallDeviceApplication() from a conduit, the application on the handheld launches with a sysAppLaunchCmdHandleSyncCallApp launch code. To handle this launch code, the handheld application must cast the command parameter block passed to PilotMain() to a SysAppLaunchCmdHandleSyncCallAppType pointer. The SysAppLaunchCmdHandleSyncCallAppType structure contains all the information that the caller passed into SyncCallDeviceApplication() on the desktop as well as the necessary fields to pass the result back to the desktop.

After the handheld application processes the sysAppLaunchCmdHandleSyncCallApp launch code, it must send a DlkCallAppReplyParamType reply back to the desktop using the DLServer's (DLServer.h) DlkControl() function.

[Table 5.1](#) and [Table 5.2](#) are some important mappings from the [CCallApplicationParams](#) class of Sync Manager on the desktop to the SysAppLaunchCmdHandleSyncCallAppType and DlkCallAppReplyParamType structures on the handheld.

**Table 5.1 Mapping from desktop Sync Manager
CCallApplicationParams to handheld
SysAppLaunchCmdHandleSyncCallAppType**

CCallApplicationParams	SysAppLaunchCmdHandleSyncCallAppType structure
m_wActionCode	action
m_dwParamSize	dwParamSize
m_pParam	paramP

**Table 5.2 Mapping from desktop Sync Manager
CCallApplicationParams to handheld
DlkCallAppReplyParamType**

CCallApplicationParams	DlkCallAppReplyParamType structure
m_dwResultBufSize	dwResultSize
m_pResultBuf	resultP
m_dwresultCode	dwresultCode

For more information and example handheld application code, see *Exploring Palm OS: System Management*.

Compatibility Sync Manager version: 2.5 or later.
 Palm OS version: Palm OS Cobalt, version 6.0.1 or later.

See Also [CCallApplicationParams](#)

Common Sync Manager API

SyncGenerateBackupFileName

SyncGenerateBackupFileName Function

Purpose Generates the unique backup filename of a database specified by its name, creator, type, and attributes.

Declared In SyncCommon.h

Prototype HSError SyncGenerateBackupFileName
(DBDatabaseInfo &*info*, TCHAR **fileName*)

Parameters → *info*

A reference to a [DBDatabaseInfo](#) structure whose name, creator, type and attributes fields specify the target database. Note that the database does *not* have to exist on the handheld.

← *fileName*

A buffer of size _MAX_FNAME or larger. Do not pass in NULL. Receives the generated backup filename.

Returns If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following negative error code values:

SYNCERR_BAD_ARG

One or more of the parameters passed in are invalid.

SYNCERR_COMM_NOT_INIT

SYNCERR_LOCAL_BUFF_TOO_SMALL

The *filename* buffer was not large enough to contain the filename.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments

Versions of Palm OS *earlier than Palm OS Cobalt* uniquely identify databases by name only. Therefore, in versions of HotSync Manager *earlier than 6.0*, the default Backup conduit creates backup filenames that consist of only the database name with an extension that depends on the database attributes (PRC for classic resource databases and PDB for classic record databases). The case of database names is incorrectly ignored in backup filenames.

However, Palm OS Cobalt uniquely identifies schema and extended databases by *name and creator ID*. Therefore the Backup conduit that ships with HotSync Manager 6.0 and later generates backup

filenames for all databases based on both their name and creator ID. This function generates such filenames.

This function generates standard backup filenames that a conduit can pass into the [SyncBackupDatabase\(\)](#) to create an image file on the desktop of a database on the handheld—though, you do not have to call this function before SyncBackupDatabase(), because SyncBackupDatabase() can generate the filename itself. Another use for this function is to determine whether a database has a backup image on the desktop already—that is, whether it has been backed up.

This function generates filenames with extensions based on the database attributes and type as shown in [Table 5.3](#).

Table 5.3 Backup filename extensions

Extension	Database description
PRC	Classic resource databases
PDB	Classic or extended record databases
SDB	Schema databases (nonsecure)
SSD	Secure schema databases
VLT	Security vault databases

The Sync Manager uses a name-mangling scheme to prevent collisions between Palm OS database names, which are case-sensitive, and Windows filenames, which are case-insensitive.

Compatibility Sync Manager version: 2.4 or later.

Palm OS version: Not applicable.

See Also

[DBDatabaseInfo](#), [SyncBackupDatabase\(\)](#),
[SyncBackupSecurityData\(\)](#),
[SyncIsDatabaseBackupNeeded\(\)](#)

Common Sync Manager API

SyncGetAPIVersion

SyncGetAPIVersion Function

Purpose	Retrieves the version of the Sync Manager API that is installed on the desktop computer.
Declared In	<code>SyncCommon.h</code>
Prototype	<code>SInt32 SyncGetAPIVersion (UInt32 *pdwMajor, UInt32 *pdwMinor)</code>
Parameters	<p>$\leftarrow pdwMajor$ The major version number of the Sync Manager API on the desktop computer. Specify NULL to ignore this value.</p> <p>$\leftarrow pdwMinor$ The minor version number of the Sync Manager API on the desktop computer. Specify NULL to ignore this value.</p>
Returns	Returns 0.
Comments	Use the API version information to determine which of the Sync Manager functions you can call on the desktop computer. For information about Sync Manager API versions and their relationship to HotSync Manager versions, see “ Supported Versions of HotSync Manager and Sync Manager ” on page 2 in <i>Introduction to Conduit Development</i> .
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.
See Also	“Versions of the Sync Manager API” on page 433

SyncGetDesktopTrustStatus Function

Purpose	Determines whether the HotSync operation in progress is with a trusted desktop.
Declared In	SyncCommon.h
Prototype	<pre>HSError SyncGetDesktopTrustStatus (eDesktopTrustStatus *pTrustStatus)</pre>
Parameters	<p>← <i>pTrustStatus</i> A pointer to an eDesktopTrustStatus value that receives the trust status of the desktop: trusted, not trusted, or trust status not verified.</p>
Returns	<p>If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following negative error code values:</p> <p>SYNCERR_BAD_ARG One or more of the parameters passed in are invalid. See “Common Sync Manager Error Codes” on page 486 for a description of all Sync Manager error codes.</p>
Comments	<p>Call this function only during a HotSync operation, because the Sync Manager must retrieve the desktop trust status from the handheld.</p> <p>This function passes back eDesktopTrustNotVerified, if it is called after the Sync Manager connects to the handheld but before the Sync Manager performs authentication.</p>
Compatibility	<p>Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.</p>
See Also	<p>eDesktopTrustStatus</p>

Common Sync Manager API

SyncGetHHOSVersion

SyncGetHHOSVersion Function

Purpose	Retrieves the version number of the operating system on the handheld.
Declared In	SyncCommon.h
Prototype	UInt16 SyncGetHHOSVersion (UInt16 * <i>pwRomVMinor</i>)
Parameters	<i>pwRomVMinor</i> The minor version number of the operating system. You can pass NULL to ignore this value.
Returns	If successful, returns the major version number of the operating system on the handheld. If unsuccessful, returns 0, which generally indicates a lost connection.
Comments	Many Sync Manager functions work only with handhelds running a minimum version of Palm OS. Before using such Sync Manager functions, you can call this function first to check the handheld's Palm OS version. Alternatively, call SyncReadSystemInfo() and then use the SYNCROMVMAJOR() and SYNCROMVMINOR() macros on the <i>m_RomSoftVersion</i> member of the CSystemInfo object. In fact, this is what SyncGetHHOSVersion() does for you.
Compatibility	Sync Manager version: 2.1 or later. Palm OS version: All.
See Also	SyncReadSystemInfo() , CSystemInfo

SyncHHToHostDWord Function

Purpose	Returns the desktop computer's representation of the specified 32-bit UInt32 value from the handheld.
Declared In	SyncCommon.h
Prototype	UInt32 SyncHHToHostDWord (UInt32 dwValue)
Parameters	$\rightarrow dwValue$ The UInt32 value from the handheld.
Returns	The UInt32 result of the conversion. This is the representation of the value on the desktop computer.
Comments	This function performs byte swapping as required and returns the converted value as the function result. For Windows, which stores multi-byte integer values in little-endian order, this conversion is required only for multi-byte values in <i>classic</i> databases. <i>Schema</i> and <i>extended</i> databases represent multi-byte integers in the same little-endian format as Windows, and therefore do not require conversion.
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.
See Also	SyncHHToHostWord() , SyncHostToHHWord() , SyncHostToHHWord()

Common Sync Manager API

SyncHHToHostWord

SyncHHToHostWord Function

Purpose	Returns the desktop computer's representation of the specified 16-bit UInt16 value from the handheld.
Declared In	SyncCommon.h
Prototype	UInt16 SyncHHToHostWord (UInt16 wValue)
Parameters	$\rightarrow wValue$ The UInt16 value from the handheld.
Returns	The WORD result of the conversion. This is the representation of the value on the desktop computer.
Comments	This function performs byte swapping as required and returns the converted value as the function result. For Windows, which stores multi-byte integer values in little-endian order, this conversion is required only for multi-byte values in <i>classic</i> databases. <i>Schema</i> and <i>extended</i> databases represent multi-byte integers in the same little-endian format as Windows, and therefore do not require conversion.
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.
See Also	SyncHHToHostDWord() , SyncHostToHHDWord() , SyncHostToHHWord()

SyncHostToHHDWord Function

Purpose	Returns the handheld's representation of the specified 32-bit UInt32 value from the desktop computer.
Declared In	SyncCommon.h
Prototype	UInt32 SyncHostToHHDWord (UInt32 dwValue)
Parameters	$\rightarrow dwValue$ The UInt32 value from the desktop computer.
Returns	This function performs byte swapping as required and returns the converted value as the function result. For Windows, which stores multi-byte integer values in little-endian order, this conversion is required only for multi-byte values in <i>classic</i> databases. <i>Schema</i> and <i>extended</i> databases represent multi-byte integers in the same little-endian format as Windows, and therefore do not require conversion.
Compatibility	Sync Manager version: 2.1 or later. Palm OS version: All.
See Also	SyncHHToHostDWord() , SyncHHToHostWord() , SyncHostToHHWord()

Common Sync Manager API

SyncHostToHHWord

SyncHostToHHWord Function

Purpose	Returns the handheld's representation of the specified 16-bit UInt16 value from the desktop computer.
Declared In	SyncCommon.h
Prototype	UInt16 SyncHostToHHWord (UInt16 wValue)
Parameters	$\rightarrow wValue$ The UInt16 value from the desktop computer.
Returns	This function performs byte swapping as required and returns the converted value as the function result. For Windows, which stores multi-byte integer values in little-endian order, this conversion is required only for multi-byte values in <i>classic</i> databases. <i>Schema</i> and <i>extended</i> databases represent multi-byte integers in the same little-endian format as Windows, and therefore do not require conversion.
Compatibility	Sync Manager version: 2.1 or later. Palm OS version: All.
See Also	SyncHHToHostDWord() , SyncHHToHostWord() , SyncHostToHHDWord()

SyncInstallAndBackupDatabase Function

Purpose Installs a database on the handheld from an image file on the desktop and then backs up the same database.

Declared In SyncCommon.h

Prototype

```
HSError SyncInstallAndBackupDatabase
    (TCHAR *filePath, TCHAR *backupPath,
     HSBool *pIsInstalled, DBDatabaseInfo *pDbInfo)
```

Parameters

→ *filePath*

A pointer to the full path and filename of the image file to install, as a null-terminated TCHAR array. Do not pass in NULL.

↔ *backupPath*

The destination path or filename of the backup file, as a null-terminated TCHAR array. If this is a directory, then the Sync Manager generates the filename automatically, appends it to the specified path, and backs up the database using this full path and filename. If the caller passes in NULL, then the default backup path is used:

<CurrentHotSyncUserFolder>\Backup. Upon return, this parameter receives the actual path and filename used.

← *pIsInstalled*

A pointer to a boolean that indicates whether the database was successfully installed.

← *pDbInfo*

A pointer to a [DBDatabaseInfo](#) structure that receives information about the database. This parameter is nullable. This structure is valid only when **pIsInstalled* is true.

Returns If successful, returns SYNCERR_NONE.

If unsuccessful, returns one of the following negative error code values:

SYNCERR_ACCESS_DENIED

Access to the specified database was denied by the handheld Authorization Manager—for example, because the database is secure and the desktop is not trusted.

SYNCERR_ARG_MISSING

SYNCERR_BAD_ARG

One or more of the parameters passed in are invalid.

Common Sync Manager API

SyncInstallAndBackupDatabase

SYNCERR_COMM_NOT_INIT

SYNCERR_DISK_FULL

SYNCERR_FILE_NOT_FOUND

SYNCERR_FILE_OPEN

SYNCERR_INVALID_IMAGE

The specified file is not a valid image of a Palm OS database.

SYNCERR_LOCAL_CANCEL_SYNC

SYNCERR_LOCAL_MEM

SYNCERR_LOST_CONNECTION

SYNCERR_NOT_FOUND

The specified image file is not present on the desktop or could not be read.

SYNCERR_OPERATION_ABORTED

SYNCERR_PATH_NOT_FOUND

SYNCERR_REMOTE_BAD_ARG

SYNCERR_REMOTE_MEM

SYNCERR_REMOTE_NO_SPACE

The handheld has insufficient memory to complete this operation.

SYNCERR_REMOTE_SYS

SYNCERR_TOO_MANY_OPEN_FILES

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments	<p>This function installs a Palm OS database on the handheld in the same way that SyncInstallDatabase() does and backs up the same database in the same way that SyncBackupDatabase() does. However, this function is significantly faster than performing the two operations separately, because the backup operation does not transfer the database back from the handheld; it simply copies the install file to the backup directory.</p> <p>This function performs the backup operation only when the database backup bit is set <i>or</i> the database creator ID does not consist entirely of lowercase letters—that is, the creator ID is not reserved for use by PalmSource, Inc.; in the latter case, the backup bit is automatically set in both the handheld database and the desktop image. When the backup is performed, the backup image's creation, modification, and backup dates are updated with the corresponding handheld values.</p> <p>This function's return value along with the output parameters indicate the success or failure of each operation. If this function returns SYNCERR_NONE, then <i>*pIsInstalled</i> is true. If this function returns SYNCERR_NONE and (<i>pDbInfo->attribues</i> & dmHdrAttrBackup) is nonzero, then the database was successfully backed up. If this function returns any other value and <i>*pIsInstalled</i> is true, then the error occurred during the backup. Otherwise, <i>*pIsInstalled</i> is false, and the function failed during installation; this implies that <i>*pDbInfo</i> is not updated.</p>
Compatibility	<p>Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.</p>
See Also	<p>SyncInstallDatabase(), SyncBackupDatabase(), SyncIsDatabaseBackupNeeded(), SyncGenerateBackupFileName(), SyncBackupSecurityData()</p>

Common Sync Manager API

SyncInstallDatabase

SyncInstallDatabase Function

Purpose	Installs a database on the handheld from an image file on the desktop.
Declared In	<code>SyncCommon.h</code>
Prototype	<code>HSError SyncInstallDatabase (TCHAR *filePath, DBDatabaseInfo *pDbInfo = 0)</code>
Parameters	<p>→ <i>filePath</i> A pointer to the full path and filename of the image file to install, as a null-terminated TCHAR array. Do not pass in NULL.</p> <p>← <i>pDbInfo</i> A pointer to a DBDatabaseInfo structure that receives information about the database. This optional parameter is nullable.</p>
Returns	<p>If successful, returns <code>SYNCERR_NONE</code>.</p> <p>If unsuccessful, returns one of the following negative error code values:</p> <p><code>SYNCERR_ACCESS_DENIED</code> Access to the specified database was denied by the handheld Authorization Manager—for example, because the database is secure and the desktop is not trusted.</p> <p><code>SYNCERR_ARG_MISSING</code></p> <p><code>SYNCERR_BAD_ARG</code> One or more of the parameters passed in are invalid.</p> <p><code>SYNCERR_COMM_NOT_INIT</code></p> <p><code>SYNCERR_FILE_NOT_FOUND</code></p> <p><code>SYNCERR_INVALID_IMAGE</code> The specified file is not a valid image of a Palm OS database.</p> <p><code>SYNCERR_LOCAL_CANCEL_SYNC</code></p> <p><code>SYNCERR_LOCAL_MEM</code></p> <p><code>SYNCERR_LOST_CONNECTION</code></p> <p><code>SYNCERR_NOT_FOUND</code> The specified image file is not present on the desktop or could not be read.</p>

SYNCERR_OPERATION_ABORTED
SYNCERR_PATH_NOT_FOUND
SYNCERR_REMOTE_BAD_ARG
SYNCERR_REMOTE_MEM
SYNCERR_REMOTE_NO_SPACE
The handheld has insufficient memory to complete this operation.
SYNCERR_REMOTE_SYS
SYNCERR_TOO_MANY_OPEN_FILES
SYNCERR_UNKNOWN_REQUEST
The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments

With this function, a conduit can write an entire database to the handheld in one call, unlike other Sync Manager functions do. A conduit can call this function at any time after HotSync Manager calls its `OpenConduit()` entry point and before it returns. Alternatively, before a HotSync operation begins, a desktop application or installer can queue a database or file to be installed by a default install conduit during the next HotSync; see [Chapter 15, “Install Aide API,”](#) on page 951 for details.

This function reads a file on the desktop that is an image of any valid Palm OS database. The Sync Manager validates the file to some extent and then transfers it as a database to memory on the handheld. If a database with the same name and creator ID in the relevant namespace already exists on the handheld, this function deletes it and writes a new database from the image file.

If the database creator ID does not consist of all lowercase letters—that is, the creator ID is not reserved for use by PalmSource, Inc.—this function automatically sets the database’s backup bit.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[SyncInstallAndBackupDatabase\(\)](#),
[SyncGenerateBackupFileName\(\)](#)

Common Sync Manager API

SyncIsDatabaseBackupNeeded

SynIsDatabaseBackupNeeded Function

Purpose Determines whether the desktop backup file for a database on the handheld is out-of-date.

Declared In SyncCommon.h

Prototype HSError SyncIsDatabaseBackupNeeded
(DBDatabaseInfo &hhInfo, TCHAR *filePath,
DBDatabaseInfo *pDtInfo, HSBool *pDbExists,
HSBool *pFileExists, HSBool *pBackupNeeded)

Parameters $\leftrightarrow hhInfo$
A reference to a [DBDatabaseInfo](#) structure that specifies the name, creator ID, type, and attributes of the handheld database to test. Receives values for all fields in section 1 of a DBDatabaseInfo structure for the matching database on the handheld, if it exists. Note that to generate the backup filename if *filePath* is a directory, this function uses the type and attributes that it passes *back*.

$\leftrightarrow filePath$
A pointer to a TCHAR buffer that specifies the desktop filename or directory of the corresponding backup file to test. Do not pass in NULL. If this parameter specifies a directory that exists and the handheld database exists, then this parameter receives the full path with the automatically generated filename appended.

$\leftarrow pDtInfo$
A pointer to a [DBDatabaseInfo](#) structure that receives values for all fields in section 1 for the desktop backup file specified by *filePath*, if it exists. The caller can pass in NULL.

$\leftarrow pDbExists$
If true, then the specified database exists on the handheld. If false, it does not exist. The caller can pass in NULL.

$\leftarrow pFileExists$
If true, then the specified backup file exists on the desktop. If false, it does not exist. The caller can pass in NULL.

$\leftarrow pBackupNeeded$
If TRUE, then the specified backup file is out-of-date compared to the specified database on the handheld. If

FALSE, the file is not out-of-date as defined in the “Comments” section. The caller can pass in NULL.

- Returns**
- If successful, returns SYNCERR_NONE.
- If unsuccessful, returns one of the following negative error code values:
- SYNCERR_ACCESS_DENIED
 - SYNCERR_ARG_MISSING
 - SYNCERR_BAD_ARG
 - One or more of the parameters passed in are invalid.
 - SYNCERR_COMM_NOT_INIT
 - SYNCERR_FILE_NOT_FOUND
 - SYNCERR_INVALID_IMAGE
 - SYNCERR_LOCAL_CANCEL_SYNC
 - SYNCERR_LOCAL_MEM
 - SYNCERR_LOST_CONNECTION
 - SYNCERR_PATH_NOT_FOUND
 - SYNCERR_REMOTE_BAD_ARG
 - SYNCERR_REMOTE_CANCEL_SYNC
 - SYNCERR_REMOTE_MEM
 - SYNCERR_REMOTE_NO_SPACE
 - SYNCERR_REMOTE_SYS
 - SYNCERR_TOO_MANY_OPEN_FILES
 - Insufficient file handles are available on the desktop.
 - SYNCERR_UNKNOWN_REQUEST
 - The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.
- See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Common Sync Manager API

SyncIsDatabaseBackupNeeded

Comments This function determines whether a desktop backup file exists for, or is older than, a corresponding database on the handheld. A conduit can call this function at any time after HotSync Manager calls its *OpenConduit()* entry point and before it returns. Note that the Sync Manager can back up secure databases only to trusted desktops.

This function considers a backup file on the desktop to be out of date (and therefore sets *pBackupNeeded* to TRUE) only if *all* of the following statements are true:

- The specified database is present on the handheld.
- The database's backup bit is set.
- One or more of the following is true:
 - The backup file does not exist on the desktop.
 - The backup file exists, but it is not a backup file for the specified database, either because it isn't of the same type or it doesn't have the same database name and creator ID.
 - The handheld and desktop creation dates differ.
 - The handheld and desktop modification dates differ.
 - The handheld last backup date is zero.

NOTE: If the database's backup bit is not set, this function always returns FALSE.

Consider when the *filePath* parameter specifies a directory that exists on the desktop and the *hhInfo* parameter specifies a database that exists on the handheld. In this case, the Sync Manager generates the filename of the backup file based on information in the handheld database header. When this function returns, it appends the filename to the original directory path and passes back this full path and filename via *filePath*.

However, consider when the *filePath* parameter specifies a directory that does *not* exist on the desktop and the specified database exists on the handheld. In this case, the Sync Manager assumes that the last part of the path is the filename of the backup file. When this function returns, it does *not* change **filePath* by appending a generated filename.

Compatibility	Sync Manager version: 2.4 or later. Palm OS version: Palm OS Cobalt, version 6.0 or later.
See Also	DBDatabaseInfo , SyncBackupDatabase() , SyncGenerateBackupFileName()

Common Sync Manager API

SyncLoopBackTest

SyncLoopBackTest Function

Purpose (*Private*) Tests communications with the device.

Declared In SyncCommon.h

Prototype SInt32 SyncLoopBackTest (UInt32 *dwSizeSend*,
 HSByte **pDataSend*, UInt32 **pdwSizeRecv*,
 HSByte **pDataRecv*)

Parameters → *dwSizeSend*
 A UInt32 value specifying the number of bytes in the
 pDataSend buffer.

→ *pDataSend*
 A pointer to a buffer containing the data to send.

↔ *pdwSizeRecv*
 A pointer to a UInt32 value that receives the number of
 bytes received.

← *pDataRecv*
 A pointer to a buffer to receive the read data.

Returns If successful, returns SYNCERR_NONE.

If unsuccessful, returns a negative error code value.

Comments This function is for internal testing purposes only.

IMPORTANT: Do not use this functions in your code. Doing so can interfere with HotSync Manager and corrupt all data on the handheld.

Compatibility Sync Manager version: 2.3 or later.
Palm OS version: All.

SyncReadBackupImageInfo Function

Purpose Reads the database header information from a backup image file on the desktop.

Declared In SyncCommon.h

Prototype HSError SyncReadBackupImageInfo (TCHAR **filePath*, DBDatabaseInfo &*info*)

Parameters → *filePath*
A pointer to the full path and filename of the backup image file, as a null-terminated TCHAR array. Do not pass in NULL.

← *info*
A pointer to a [DBDatabaseInfo](#) structure that receives values for all fields in section 1 for the backup image file specified by *filePath*, if it exists.

Returns If successful, returns SYNCERR_NONE.
If unsuccessful, returns one of the following negative error code values:
SYNCERR_ACCESS_DENIED
SYNCERR_BAD_ARG
One or more of the parameters passed in are invalid.
SYNCERR_COMM_NOT_INIT
SYNCERR_FILE_NOT_FOUND
SYNCERR_INVALID_IMAGE
The specified file is not a valid image of a Palm OS database.
SYNCERR_NOT_FOUND
SYNCERR_PATH_NOT_FOUND
The path specified by *filePath* does not exist.
SYNCERR_TOO_MANY_OPEN_FILES
Insufficient file handles are available on the desktop.
SYNCERR_UNKNOWN
See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Common Sync Manager API

SyncReadBackupImageInfo

Comments This function can be called outside of a HotSync operation.

Compatibility Sync Manager version: 2.4 or later.
Palm OS version: Not applicable.

See Also [SyncBackupDatabase\(\)](#), [SyncInstallDatabase\(\)](#)

SyncReadFeature Function

Purpose Retrieves a 32-bit feature value from the Feature Manager on the handheld.

Declared In SyncCommon.h

Prototype `SInt32 SyncReadFeature (UInt32 dwFtrCreator,
 UInt16 wFtrNum, UInt32 *pdwFtrValue)`

Parameters → *dwFtrCreator*
The ID of the feature creator. This is usually the creator ID of the application that publishes the feature.

→ *wFtrNum*
The feature number. Distinguishes between different features of a particular creator.

← *pdwFtrValue*
The retrieved value of the specified feature.

Returns If successful, returns 0, which means that the feature was retrieved.
If unsuccessful, returns one of the following nonzero error code values:

SYNCERR_COMM_NOT_INIT

SYNCERR_LOST_CONNECTION

SYNCERR_REMOTE_SYS

SYNCERR_REMOTE_MEM

SYNCERR_UNKNOWN_REQUEST

SYNCERR_NOT_FOUND

The requested feature could not be found, which indicates that it is not registered.

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Common Sync Manager API

SyncReadFeature

Comments	Features are stored in volatile storage that is erased and re-initialized during system reset. Palm OS and applications can register features using their own creator IDs. The contents of features are completely application-specific. For more information about the Feature Manager, see the <i>Exploring Palm OS: System Management</i> .
Compatibility	Sync Manager version: 2.1 or later. Palm OS version: 2.0 or later.

SyncReadSingleCardInfo Function

Purpose	Retrieves information about a memory card on the handheld.
Declared In	SyncCommon.h
Prototype	SInt32 SyncReadSingleCardInfo (CCardInfo &rInfo)
Parameters	$\leftrightarrow rInfo$ An object of the CCardInfo class, which specifies and receives information about a memory card. The caller must specify the m_CardNo member. This function passes back values in the other members.
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS SYNCERR_REMOTE_MEM SYNCERR_NOT_FOUND The specified card number is outside of the range of available cards. See " Common Sync Manager Error Codes " on page 486 for a description of all Sync Manager error codes.
Comments	You can use this function to determine the total number of ROM and RAM-based databases prior to calling the SyncReadDBList() function. The value passed back in the m_FreeRam field depends on the version of Palm OS running on the handheld: <ul style="list-style-type: none">if the handheld is running version 3.0 or later, the value of m_FreeRam includes unused RAM in storage heaps only, which is the amount of memory available for records or resources.if the handheld is running an earlier version of Palm OS, the value of m_FreeRam is the sum of unused RAM in both storage and dynamic heaps. However, you can only store data in the storage heaps.

Common Sync Manager API

SyncReadSingleCardInfo

Note that in Palm OS Cobalt, the concept of a memory card no longer exists. For the `m_CardNo` member, always specify 0, unless you need this function to work for earlier versions of Palm OS on certain HandSpring handhelds, which supported more than one memory card.

Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All. See the “Comments” section for version-based behavior differences.
See Also	CCardInfo , SyncReadDBList()

SyncReadSysDateTime Function

Purpose Retrieves the current date and time, according to the system clock on the handheld.

Declared In SyncCommon.h

Prototype SInt32 SyncReadSysDateTime (SInt32 &rDate)

Parameters ← rDate
The system date and time on the handheld, as a `time_t` value. If an error occurs, the value of `rDate` is either -1 or 0.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

SYNCERR_COMM_NOT_INIT

SYNCERR_LOST_CONNECTION

SYNCERR_REMOTE_SYS

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments For handhelds running Palm OS Cobalt, `rDate` is expressed as a UTC value; otherwise, it is expressed in the handheld’s local time.

Compatibility Sync Manager version: 2.0 or later.
Palm OS version: All.

See Also [SyncWriteSysDateTime \(\)](#)

Common Sync Manager API

SyncReadSystemInfo

SyncReadSystemInfo Function

Purpose	Retrieves the Palm OS software version and product information for the handheld.
Declared In	SyncCommon.h
Prototype	SInt32 SyncReadSystemInfo (CSysInfo &rInfo)
Parameters	$\leftrightarrow rInfo$ An object of the CSysInfo class, which specifies and receives information about the handheld. The caller must specify the m_AllocedLen.
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS See “ Common Sync Manager Error Codes ” on page 486 for a description of all Sync Manager error codes.
Comments	To use this function, allocate the m_ProductIDText buffer in an object of the CSysInfo class and fill in the m_AllocedLen member of the object. You must fill in m_AllocedLen with the size of the buffer that you allocated and assigned to m_ProductIDText. This function stores the handheld system information into the buffer. The retrieved product ID is a binary (non-ASCII) sequence of bytes. Currently, all handhelds return the same four bytes: 0x00, 0x01, 0x00, 0x00. This function also fills in the m_RomSoftVersion, m_LocalId, and m_ProdIdLength fields.
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.
See Also	CSysInfo , SyncGetHHOSVersion()

SyncReadUserID Function

Purpose Retrieves information about the user of the handheld, including the user name, last synchronization date, and encrypted password.

Declared In SyncCommon.h

Prototype SInt32 SyncReadUserID (CUserIdInfo &rInfo)

Parameters ← rInfo

An object of the [CUserIdInfo](#) class, which receives information about the user of the handheld.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

SYNCERR_COMM_NOT_INIT

SYNCERR_LOST_CONNECTION

SYNCERR_REMOTE_SYS

SYNCERR_REMOTE_MEM

See “[Common Sync Manager Error Codes](#)” on page 486 for a description of all Sync Manager error codes.

Comments User information is written to a new or reset handheld after completion of a synchronization. If this has not yet occurred on the handheld, the user information is empty, and the m_NameLength member of rInfo is set to 0.

IMPORTANT: When you call `SyncReadUserID()` to read information from a handheld that is running a version of Palm OS earlier than version 3.0, the `LastSyncDate` member of the `CUserIdInfo` structure is set to 0. The `LastSyncDate` member is correctly set to the most recent synchronization date for version 3.0 and later.

Compatibility Sync Manager version: 2.0 or later.
Palm OS version: All.
See the “Comments” section for version-based behavior differences.

See Also [CUserIdInfo](#), [SyncReadSystemInfo\(\)](#)

Common Sync Manager API

SyncRebootSystem

SyncRebootSystem Function

Purpose	Sends a request to soft-reset the handheld at the end of the HotSync operation.
Declared In	SyncCommon.h
Prototype	SInt32 SyncRebootSystem (void)
Parameters	None.
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_SYS See " Common Sync Manager Error Codes " on page 486 for a description of all Sync Manager error codes.
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.

SyncRegisterConduit Function

Purpose	Registers a conduit that is about to start synchronization operations and returns a handle for use with other Sync Manager functions.
Declared In	SyncCommon.h
Prototype	SInt32 SyncRegisterConduit (CONDHANDLE &rHandle)
Parameters	$\leftarrow rHandle$ The address of the handle to your conduit.
Returns	If successful, returns 0, which means that the conduit is registered with Sync Manager and can proceed with synchronization. If the conduit could not be registered, returns -1, which means that the previous conduit did not unregister itself and you cannot proceed with synchronization. If unsuccessful for another reason, returns one of the following nonzero error code values: SYNCERR_COMM_NOT_INIT SYNCERR_LOST_CONNECTION SYNCERR_REMOTE_CANCEL_SYNC SYNCERR_LOCAL_CANCEL_SYNC See “ Common Sync Manager Error Codes ” on page 486 for a description of all Sync Manager error codes.
Comments	Your conduit must call SyncRegisterConduit() to register itself before making any other calls to the Sync Manager. If SyncRegisterConduit() succeeds, your conduit must call SyncUnRegisterConduit() before it returns from its OpenConduit() entry point.
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.
See Also	SyncUnRegisterConduit()

Common Sync Manager API

SyncRestoreSecurityData

SyncRestoreSecurityData Function

Purpose	Restores security vault databases from the desktop to the handheld.
Declared In	SyncCommon.h
Prototype	HSError SyncRestoreSecurityData (TCHAR *dir)
Parameters	$\rightarrow dir$ A pointer to a null-terminated TCHAR array that specifies the directory that holds image files of vault databases to restore. Do not pass in NULL. This directory must contain one or more vault databases previously backed up by a call to SyncBackupSecurityData() . It can also contain other files, but only if they do not use the same filename extension as vaults.
Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following negative error code values: SYNCERR_ACCESS_DENIED SYNCERR_ARG_MISSING SYNCERR_BAD_ARG The <i>dir</i> parameter is NULL. SYNCERR_COMM_NOT_INIT SYNCERR_FILE_NOT_FOUND SYNCERR_INVALID_IMAGE One or more files in the specified directory is not a valid image of a Palm OS vault database. SYNCERR_LOCAL_CANCEL_SYNC SYNCERR_LOCAL_MEM SYNCERR_LOST_CONNECTION SYNCERR_NOT_FOUND No vault databases are present on the desktop or could not be read. SYNCERR_OPERATION_ABORTED SYNCERR_PATH_NOT_FOUND SYNCERR_REMOTE_BAD_ARG

SYNCERR_REMOTE_MEM

SYNCERR_REMOTE_NO_SPACE

The handheld has insufficient memory to complete this operation.

SYNCERR_REMOTE_PASSWORD

The restore operation failed because of an invalid password.

SYNCERR_REMOTE_SYS

SYNCERR_TOO_MANY_OPEN_FILES

SYNCERR_UNKNOWN

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

See "[Common Sync Manager Error Codes](#)" on page 486 for a description of all Sync Manager error codes.

Comments

After a handheld is hard-reset, [vaults](#) must be restored to the handheld *before* all other databases and in a specific order so that the handheld Authorization Manager allows other secure databases to be restored afterwards. This function restores all vault databases from the specified directory and in the order mandated by the Authorization Manager. If the vault already exists on the handheld, this function deletes it from the handheld and restores the vault from the desktop.

Compatibility

Sync Manager version: 2.4 or later.

Palm OS version: Palm OS Cobalt, version 6.0 or later.

See Also

[SyncBackupSecurityData\(\)](#)

Common Sync Manager API

SYNCROMVMAJOR

SYNCROMVMAJOR Macro

Purpose	Decodes the major version number of Palm OS on the handheld.
Declared In	SyncCommon.h
Prototype	#define SYNCROMVMAJOR (1)
Parameters	$\rightarrow 1$ The version number received by the <code>m_RomSoftVersion</code> member of the CSystemInfo object that is passed back by SyncReadSystemInfo() .
Returns	The major version number as a <code>UInt16</code> value.
Comments	To use this macro, pass in the value of the <code>m_RomSoftVersion</code> member of the CSystemInfo object that is passed back by SyncReadSystemInfo() . The value of <code>m_RomSoftVersion</code> is of the form <code>0xMMmfssbb</code> , where <ul style="list-style-type: none">• MM = major version• m = minor version• f = bug fix• s = stage (3 = release, 2 = beta, 1 = alpha, 0 = development)• bbb = build number for unreleased versions For example, if <code>m_RomSoftVersion</code> = 0x01522003, then <code>SYNCROMVMAJOR()</code> returns 0x01.
See Also	SyncReadSystemInfo() , CSystemInfo , SYNCROMVMINOR()

SYNCROMVMINOR Macro

Purpose	Decodes the minor version number of Palm OS on the handheld.
Declared In	SyncCommon.h
Prototype	#define SYNCROMVMINOR (1)
Parameters	→ 1 The version number received by the m_RomSoftVersion member of the CSystemInfo object that is passed back by SyncReadSystemInfo() .
Returns	The minor version number as a UInt16 value.
Comments	To use this macro, pass in the value of the m_RomSoftVersion member of the CSystemInfo object that is passed back by SyncReadSystemInfo() . The value of m_RomSoftVersion is of the form 0xMMmfssbbb, where <ul style="list-style-type: none">• MM = major version• m = minor version• f = bug fix• s = stage (3 = release, 2 = beta, 1 = alpha, 0 = development)• bbb = build number for unreleased versions For example, if m_RomSoftVersion = 0x01522003, then SYNCROMVMINOR () returns 0x5.
See Also	SyncReadSystemInfo() , CSystemInfo , SYNCROMMAJOR()

Common Sync Manager API

SyncUnRegisterConduit

SyncUnRegisterConduit Function

Purpose	Unregisters a conduit that was successfully registered with a previous call to SyncRegisterConduit() , and cleans up any system resources that the Sync Manager allocated for the conduit.
Declared In	<code>SyncCommon.h</code>
Prototype	<code>SInt32 SyncUnRegisterConduit (CONDHANDLE handle)</code>
Parameters	<code>→ handle</code> A handle to a conduit that was returned from a previous call to the SyncRegisterConduit() function.
Returns	If successful, returns 0, which means that your conduit was successfully unregistered. If <code>handle</code> is not a valid conduit handle, returns -1. If unsuccessful for another reason, returns the following nonzero error code value: <code>SYNCERR_COMM_NOT_INIT</code> See “ Common Sync Manager Error Codes ” on page 486 for a description of all Sync Manager error codes.
Comments	This function determines whether the handle is to a registered conduit, and if so, unregisters the conduit. If <code>handle</code> is not a handle to a registered conduit, this function returns -1.
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.
See Also	SyncRegisterConduit()

SyncWriteSysDateTime Function

Purpose	Sets the system date and time on the handheld.
Declared In	<code>SyncCommon.h</code>
Prototype	<code>SInt32 SyncWriteSysDateTime (SInt32 lDate)</code>
Parameters	<p><code>→ lDate</code> A <code>time_t</code> value that specifies the system date and time value. This value must be in the format returned by the <code>time()</code> function.</p>
Returns	<p>If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values:</p> <p><code>SYNCERR_COMM_NOT_INIT</code> <code>SYNCERR_LOST_CONNECTION</code> <code>SYNCERR_REMOTE_SYS</code> <code>SYNCERR_REMOTE_BAD_ARG</code></p> <p>If you call this function in a version of the Sync Manager API earlier than version 2.2, this function returns a <code>SYNCERR_REMOTE_BAD_ARG</code> error code.</p> <p>See “Common Sync Manager Error Codes” on page 486 for a description of all Sync Manager error codes.</p>
Comments	<p>For handhelds running Palm OS Cobalt, <code>lDate</code> is expressed as a UTC value; otherwise, it is expressed in the handheld’s local time.</p> <p>In general, conduits should avoid changing the system date and time, because this function does not notify applications on the handheld that it has changed the time. Some applications, such as PalmSource’s Date Book, need to know when the system time changes so that they can adjust their alarm settings. To work around this problem, you need to call the <code>SyncRebootSystem()</code> function, which causes a soft-reset of the handheld after the HotSync operation completes. All applications on the handheld are notified of the reset and can make any necessary adjustments.</p>

Common Sync Manager API

SyncWriteSysDateTime

IMPORTANT: Although this function is available in versions 2.0 and later of the Sync Manager API, it works properly only in Sync Manager version 2.4 and later with handhelds running Palm OS Cobalt. In versions 2.4 and later, it crashes if the handheld is running a version of Palm OS earlier than Palm OS Cobalt.

In Sync Manager API versions 2.1 and earlier, `SyncWriteSysDateTime()` returns an error code. However, in versions 2.2 and 2.3 it returns successfully, but does not actually change the time on the handheld.

Compatibility

Sync Manager version: 2.0 or later.

Palm OS version: All.

See note in Comments for version differences.

See Also

[SyncReadSysDateTime\(\)](#)

SyncYieldCycles Function

Purpose	Causes HotSync Manager to update its progress display indicator.
Declared In	SyncCommon.h
Prototype	SInt32 SyncYieldCycles (UInt16 wMaxMiliSecs)
Parameters	$\rightarrow wMaxMiliSecs$ The maximum number of milliseconds for HotSync Manager to spend servicing events. This value is currently ignored; specify a value of 1.
Returns	If successful, returns 0. If unsuccessful, returns the following nonzero error code value: SYNCERR_COMM_NOT_INIT See " Common Sync Manager Error Codes " on page 486 for a description of all Sync Manager error codes.
Comments	Call SyncYieldCycles () periodically to keep the progress display indicator current in the HotSync Progress dialog box. If you neglect to call this function frequently enough, the user interface will appear to be frozen.
Compatibility	Sync Manager version: 2.0 or later. Palm OS version: All.

Common Sync Manager API

Common Sync Manager Error Codes

Common Sync Manager Error Codes

[Table 5.4](#) lists the values of error codes that schema, extended, and classic Sync Manager functions can return. The description of each function states which errors each function can return.

All of the named error codes below are defined as preprocessor constants, which are declared in the SyncCommon.h header file.

Table 5.4 Common Sync Manager error codes

Value	Error Code	Description
0x00000000	SYNCERR_NONE	The function call succeeded without error.
0x00001000	COND_ERR_CLASS	No function returns this value. It is the offset for all other error codes returned by conduits to HotSync Manager.
0x00002000	TRANS_ERR_CLASS	No function returns this value. It is the offset for all error codes related to transport/communications with the handheld.
0x00002900	EXPAPI_ERR_CLASS	No function returns this value. It is the offset for all error codes returned by the Expansion Manager API.
0x00002A00	VFSAPI_ERR_CLASS	No function returns this value. It is the offset for all error codes returned by the VFS Manager API.
0x00004000	SYNC_ERR_CLASS	No function returns this value. It is the offset for all error codes returned by the Sync Manager API.

Table 5.4 Common Sync Manager error codes (*continued*)

Value	Error Code	Description
0x00004001	SYNCERR_UNKNOWN	An unknown error occurred. This error typically indicates the occurrence of an unexpected system error on the desktop, but it may indicate the desktop's failure to recognize a protocol error (the latter should not happen in practice).
0x00004002	SYNCERR_MORE	Returned by schema and extended Sync Manager functions. Indicates that the caller must allocate more memory to receive the complete response.
0x00004003	SYNCERR_NOT_FOUND, SYNCERR_FILE_NOT_FOUND	The requested database, record, resource, etc. cannot be found. This error code replaces the earlier SYNCERR_FILE_NOT_FOUND error code. Note: This result code is also returned when iterating through a database to indicate that there are no more records to retrieve.
0x00004004	SYNCERR_FILE_NOT_OPEN	The attempt to open the database failed.
0x00004005	SYNCERR_FILE_OPEN	Not used.
0x00004006	SYNCERR_RECORD_BUSY	The requested record is in use by someone else and will remain so indefinitely.

Common Sync Manager API

Common Sync Manager Error Codes

Table 5.4 Common Sync Manager error codes (*continued*)

Value	Error Code	Description
0x00004007	SYNCERR_RECORD_DELETED	The requested record has either been deleted or archived.
0x00004009	SYNCERR_READ_ONLY, SYNCERR_ROM_BASED	A function attempted to write to a database that either has been opened in read-only mode or is in ROM. This error code replaces the earlier SYNCERR_ROM_BASED error code.
0x0000400A	SYNCERR_COMM_NOT_INIT	An internal error code that indicates communications have not been initialized. This error occurs when a function that requires handheld access is called outside of a HotSync operation.
0x0000400B	SYNCERR_FILE_ALREADY_EXIST	The specified object could not be created on the handheld because another object with the same name already exists. For example, a database with the same name already exists.
0x0000400C	SYNCERR_FILE_ALREADY_OPEN	The specified database is already open. The function you called requires that the database be closed.
0x0000400D	SYNCERR_NO_FILES_OPEN	An read or write was attempted on a database, but the specified database is not open. This typically indicates that the input database handle is invalid.

Table 5.4 Common Sync Manager error codes (*continued*)

Value	Error Code	Description
0x0000400E	SYNCERR_BAD_OPERATION	The requested operation is not supported for the given database type—for example, an extended database write has been attempted with a schema database handle.
0x0000400F	SYNCERR_REMOTE_BAD_ARG	The handheld found that an invalid parameter has been passed to it that the desktop Sync Manager cannot detect—for example, a nonzero database handle does not correspond to any open database.
0x00004010	SYNCERR_BAD_ARG_WRAPPER	An internal Desktop Link error that indicates a protocol implementation error.
0x00004011	SYNCERR_ARG_MISSING	An internal error that indicates a miscommunication between the desktop and handheld. This error should not occur in practice.
0x00004012	SYNCERR_LOCAL_BUFF_TOO_SMALL	The passed buffer is too small for the reply data.
0x00004013	SYNCERR_REMOTE_MEM	There is insufficient memory on the handheld to receive or complete the request.

Common Sync Manager API

Common Sync Manager Error Codes

Table 5.4 Common Sync Manager error codes (*continued*)

Value	Error Code	Description
0x00004014	SYNCERR_REMOTE_NO_SPACE	There is insufficient memory in the data store on the handheld to complete the request. This generally occurs when attempting to write a record or resource to a handheld database. This error is identical to SYNCERR_REMOTE_MEM.
0x00004015	SYNCERR_REMOTE_SYS	A generic system error on the handheld. This is returned when the handheld error code is not mapped to an existing desktop error code.
0x00004016	SYNCERR_LOCAL_MEM	A memory allocation error occurred on the desktop computer—for example, when the desktop runs out of memory.
0x00004017	SYNCERR_BAD_ARG	The desktop Sync Manager found that an invalid parameter has been passed to it—for example, a database handle with a value of zero.
0x00004018	SYNCERR_LIMIT_EXCEEDED	A data limit has been exceeded on the handheld. For example, an extended Sync Manager function is used to write more than 64 KB of data to a classic database, or the HotSync error log size limit has been exceeded on the handheld.

Table 5.4 Common Sync Manager error codes (*continued*)

Value	Error Code	Description
0x00004019	SYNCERR_UNKNOWN_REQUEST	The handheld is running an unsupported version of the HotSync protocol—for example, when a function that works only with Palm OS Cobalt is called against a handheld running an earlier version of Palm OS.
0x0000401A	SYNCERR_ACCESS_DENIED	The Authorization Manager on the handheld denied access to the database, or the database cannot be unencrypted, or you cannot delete a schema database table that contains nondefault sort indexes.
0x0000401B	SYNCERR_DEVICE_NOT_CONNECTED	Function cannot complete without the device connected.
0x0000401C	SYNCERR_INVALID_IMAGE	The image being installed is not a valid PalmOS database.
0x0000401D	SYNCERR_PATH_NOT_FOUND	The directory or file path is not valid.
0x0000401F	SYNCERR_DISK_FULL	A write to the desktop disk failed because the disk is full.
0x00008000	HSAPP_ERR_CLASS	No function returns this value. It is the offset for all error codes returned by the HotSync Manager application.
0x10000000	SYNC_FATAL_ERR_MASK	No function returns this value. It is a mask to use to determine whether an error code is a fatal error.

Common Sync Manager API

Common Sync Manager Error Codes

Table 5.4 Common Sync Manager error codes (*continued*)

Value	Error Code	Description
0x10004000	SYNC_FATAL_ERR	No function returns this value. It is the offset for all error codes that indicate a fatal error.
0x10004403	SYNCERR_TOO_MANY_OPEN_FILES, SYNCERR_TOO_MANY_FILES	Request failed because there are too many open classic databases (only one classic database can be open at a time). This error code replaces the earlier SYNCERR_TOO_MANY_FILES error code.
0x10004405	SYNCERR_REMOTE_CANCEL_SYNC	The handheld detected a request to cancel the HotSync operation on the handheld.
0x10004411	SYNCERR_LOCAL_CANCEL_SYNC	The desktop Sync Manager detected a request to cancel the HotSync operation.
0x10006410	SYNCERR_LOST_CONNECTION	The connection with the handheld was lost—for example, because the handheld was removed from the cradle during a HotSync operation.



Part II

Conduit APIs

This part describes the conduit entry points and the other APIs that only conduits can call during a HotSync operation:

<u>Conduit Entry Point API</u>	495
<u>HotSync Log API</u>	529
<u>Palm OS Common Language API</u>	539
<u>Expansion Manager API</u>	547
<u>Virtual File System Manager API</u>	561

6

Conduit Entry Point API

Conduits must implement the required conduit entry point functions described in this chapter. HotSync® Manager calls these entry points to start a conduit, pass it information, allow the user to configure its operation, or retrieve information from a conduit.

The conduit entry points marked *Required* must be implemented in your conduit DLL for HotSync Manager to call. Others are optional or required only if you want to implement specific features. These entry point functions are declared in CondAPI.h. Some other elements used by these functions are declared in SyncMgr.h and Subscribe.h.

The sections in this chapter are:

Conduit Entry Point API Structures	496
Conduit Entry Point API Constants	502
Conduit-Defined Entry Point API Functions	510
HotSync Manager Callback Function	523
Conduit Entry Point API Error Codes	524

The implementation of conduit entry points is discussed in [Chapter 3, “Implementing Conduit Entry Points,”](#) on page 17 in the C/C++ Sync Suite Companion.

Conduit Entry Point API Structures

This section describes the following data structures that you use with the Conduit Entry Point API.

<u>CfgConduitInfoType</u>	Specifies additional information to a conduit when HotSync Manager calls the conduit's CfgConduit() entry point.
<u>ConduitRequestInfoType</u>	Specifies additional information to a conduit when HotSync Manager calls the conduit's GetConduitInfo() entry point to request the conduit's name.
<u>RegistrationInfoType</u>	Specifies registration information about a conduit when HotSync Manager calls a folder-registered conduit's GetConduitInfo() entry point.

CfgConduitInfoType Struct

Purpose Specifies additional information to a conduit when HotSync Manager calls the conduit's [CfgConduit\(\)](#) entry point.

Declared In CondAPI.h

Prototype

```
typedef struct CfgConduitInfoType {
    UInt32 dwVersion;
    UInt32 dwSize;
    UInt32 dwCreatorId;
    UInt32 dwUserId;
    TCHAR szUser[64];
    char m_PathName[BIG_PATH];
    eSyncTypes syncPermanent;
    eSyncTypes syncTemporary;
    eSyncTypes syncNew;
    eSyncPref syncPref;
} CFGCONDUITINFO
```

Fields dwVersion
Specifies the version number of this structure.

dwSize
Specifies the size of this structure in bytes.

dwCreatorId
Specifies the creator ID of the conduit.

dwUserId
Specifies the current HotSync user ID.

szUser
Specifies the current HotSync user name.

m_PathName
The fully-qualified path of the conduit.

syncPermanent
Receives the user's stored [permanent synchronization preference](#) for the specified conduit. This is one of the [eSyncTypes](#) enum values.

syncTemporary
Receives the user's stored [temporary synchronization preference](#) for the specified conduit. This is one of the [eSyncTypes](#) enum values.

Conduit Entry Point API

CfgConduitInfoType

syncNew

Specifies the user's new or changed synchronization preferences for HotSync Manager to store. Your conduit must specify one of the the [eSyncTypes](#) enum values.

syncPref

Specifies how HotSync Manager should store the preference that your conduit specifies in the syncNew field, either as a permanent or temporary preference. Your conduit must specify one of the the [eSyncPref](#) enum values.

Compatibility HotSync Manager version: All.
Palm OS version: All.

See Also [CfgConduit\(\)](#)

ConduitRequestInfoType Struct

Purpose	Specifies additional information to a conduit when HotSync Manager calls the conduit's GetConduitInfo() entry point to request the conduit's name.
Declared In	CondAPI.h
Prototype	<pre>typedef struct ConduitRequestInfoType { UInt32 dwVersion; UInt32 dwSize; UInt32 dwCreatorId; UInt32 dwUserId; TCHAR szUser[64]; } CONDUITREQUESTINFO</pre>
Fields	<p>dwVersion The version number of this structure.</p> <p>dwSize The size of this structure in bytes.</p> <p>dwCreatorId The creator ID of the conduit that HotSync Manager is calling.</p> <p>dwUserId The user ID of the user who started the current HotSync session.</p> <p>szUser The name of the user who started the current HotSync session.</p>
Comments	HotSync Manager passes in a structure of this type in the <i>pInArgs</i> parameter of a conduit's GetConduitInfo() entry point when it passes in a value of eConduitName in the <i>infoType</i> parameter. An example of when a conduit might need this information is when a single conduit is registered for multiple creator IDs and needs to pass back a different conduit name for HotSync Manager to display depending on the information passed to the conduit in this structure.
Compatibility	HotSync Manager version: All. Palm OS version: All.
See Also	GetConduitInfo()

Conduit Entry Point API

RegistrationInfoType

RegistrationInfoType Struct

Purpose Specifies registration information about a conduit when HotSync Manager calls a folder-registered conduit's [GetConduitInfo\(\)](#) entry point.

Declared In CondAPI.h

Prototype

```
typedef struct RegistrationInfoType {
    UInt32 dwCreatorID;
    UInt32 dwPriority;
    TCHAR szLocalDirectory[255];
    TCHAR szLocalFile[255];
    TCHAR szRemoteDB[32];
    TCHAR szTitle[255];
} REGISTRATIONINFO
```

Fields

dwCreatorID
The creator ID of the application on the handheld your conduit is responsible for synchronizing. This value is the unique key by which HotSync Manager identifies your conduit, so only one conduit can be registered with a particular creator ID at a time.

dwPriority
Execution priority for this conduit. This value is in the range 0 to 4. If no value is specified, then HotSync Manager uses a default value of 2. HotSync Manager runs conduits with a value of 0 first and those with 4 last.

szLocalDirectory
Conduit's directory name. This is the name of a subdirectory in the user's directory on the desktop computer (not a fully qualified path). Within each user's directory, each conduit can have a directory for file storage.

szLocalFile
Name of the desktop file that your conduit synchronizes with the handheld database. Your conduit can synchronize with more than one file, however. Note that this configuration entry can be either a full path and filename, or only a filename. If the value is only a filename, the file can be found in the directory specified by the **szLocalDirectory** field.

szRemoteDB
Name of a handheld database to be accessed by this conduit. This optional value can be used by conduits that are not

hard-coded with specific database names. This value is passed to the conduit to enable it to open the database on the handheld. A conduit can also use this name to create the database on the handheld if the database did not exist before synchronization. Database names are case-sensitive.

szTitle

Display name of the conduit. HotSync Manager displays this string as the name of the conduit in its user interface—for example, in the **Custom** dialog box. If you do not set this entry, HotSync Manager shows the name that your conduit provides when called by `GetConduitInfo()` or `GetConduitName()`.

Compatibility

HotSync Manager version: 6.0 and later.
Palm OS version: All.

See Also

[GetConduitInfo\(\)](#)

Conduit Entry Point API Constants

This section describes the following enumerated and preprocessor constants that you use with the Conduit Entry Point API.

ConduitCfgEnum	Indicates the version of <code>CfgConduit()</code> that HotSync Manager is calling.
ConduitInfoEnum	Indicates the type of information that HotSync Manager is requesting when it calls a conduit's <code>GetConduitInfo()</code> entry point.
Conduit Priorities	Defines the highest, lowest, and default priorities for running a conduit.
MFC Versions	Indicate whether your conduit uses MFC or not, and if so, what version.
Structure Sizes for Conduit Entry Points	Define the sizes of various structures used by some conduit entry points.
Structure Versions for Conduit Entry Points	Indicate the version numbers of certain data structures used by some conduit entry points.

ConduitCfgEnum Enum

Purpose Indicates the version of [CfgConduit\(\)](#) that HotSync Manager is calling.

Declared In CondAPI.h

Constants

eConduitCfgDoNotUse = 0xffffffffffff	HotSync Manager never passes in this value. It serves only as the upper limit of this enum.
--------------------------------------	---

eConfig1 = 0	Version 1 of the CfgConduit() entry point.
--------------	--

Compatibility HotSync Manager version: All.
Palm OS version: All.

See Also [CfgConduit\(\)](#)

Conduit Entry Point API

ConduitInfoEnum

ConduitInfoEnum Enum

Purpose	Indicates the type of information that HotSync Manager is requesting when it calls a conduit's GetConduitInfo() entry point.
Declared In	CondAPI.h
Constants	<p>eConduitName = 0 HotSync Manager requests the display name of your conduit. HotSync Manager uses the name a conduit passes back only when the conduit is not registered with a name—that is, if the Name conduit configuration entry is not set.</p> <p>eMfcVersion HotSync Manager requests whether your conduit uses MFC or not, and if so, what version. This version number is actually the version of Visual C++ that MFC shipped with, not necessarily the version number of MFC itself. Note that this enum value is deprecated in Sync Manager versions 2.4 and later. The corresponding versions of HotSync Manager do not query conduits for an MFC version. A conduit must return one of the values defined in “MFC Versions” on page 507 only if version 2.3 or earlier of Sync Manager is present.</p> <p>eDefaultAction HotSync Manager requests the type of default action performed by your conduit. A conduit must pass back one of the eSyncTypes enum values.</p> <p>eRegistrationInfo HotSync Manager requests conduit registration information when your conduit is registered via folder. If a conduit is registered via folder, it must pass back a pointer to a filled-in RegistrationInfoType structure. Note that HotSync Manager can pass in this enum value only if Sync Manager version 2.4 or later is present.</p> <p>eDoNotDisplayInConduitListForUser HotSync Manager requests whether your conduit should be displayed in the HotSync Manager Custom dialog box. If your conduit passes back a zero value or no value, then your conduit's name appears in the Custom dialog box. If your</p>

conduit passes back any nonzero value, then its name does not appear.

Note that HotSync Manager can pass in this enum value only if Sync Manager version 2.4 or later is present.

eRunAlways

HotSync Manager requests whether it should run your conduit regardless of whether an application with the same creator ID is on the handheld. If your conduit passes back a zero value or no value, then HotSync Manager runs your conduit only if an application with the same creator ID is on the handheld. If your conduit passes back any nonzero value, then it runs your conduit always.

Note that HotSync Manager can pass in this enum value only if Sync Manager version 2.4 or later is present.

eDoNotDisplayProgress

HotSync Manager requests whether it should display your conduit's name in the **HotSync Progress** dialog box during a HotSync operation. If your conduit passes back a zero value or no value, then HotSync Manager displays your conduit's name. If your conduit passes back any nonzero value, then it does not display your conduit's name.

Note that HotSync Manager can pass in this enum value only if Sync Manager version 2.4 or later is present.

eConduitInfoDoNotUse = 0xffffffff

HotSync Manager never passes in this value. It serves only as the upper limit of this enum.

Compatibility HotSync Manager version: All, except where noted above.
Palm OS version: All.

See Also [GetConduitInfo\(\)](#)

Conduit Entry Point API

Conduit Priorities

Conduit Priorities

Purpose	Defines the highest, lowest, and default priorities for running a conduit.
Declared In	CondAPI.h
Constants	<pre>#define PALM_DEF_PRIORITY 2 The priority for running a conduit if no priority value is specified. #define PALM_MAX_PRIORITY 4 HotSync Manager runs conduits with this priority <i>last</i>. #define PALM_MIN_PRIORITY 0 HotSync Manager runs conduits with this priority <i>first</i>.</pre>
Compatibility	HotSync Manager version: 6.0 and later Palm OS version: All.

MFC Versions

Purpose Indicate whether your conduit uses MFC or not, and if so, what version.

Declared In CondAPI.h

Constants

```
#define MFC_NOT_USED 0x10000000
The conduit does not use MFC.
```

```
#define MFC_VERSION_41 0x000000410
The conduit uses version 4.1.
```

```
#define MFC_VERSION_50 0x000000500
The conduit uses version 5.0.
```

```
#define MFC_VERSION_60 0x000000600
The conduit uses version 6.0.
```

Comments The version numbers correspond to the version of Visual C++ that MFC shipped with, not necessarily the version number of MFC itself.

NOTE: These constants are deprecated in HotSync Manager versions 6.0 and later, which do not query conduits for an MFC version.

Compatibility HotSync Manager version: All earlier than 6.0.
Palm OS version: All.

See Also [GetConduitInfo\(\)](#)

Conduit Entry Point API

Structure Sizes for Conduit Entry Points

Structure Sizes for Conduit Entry Points

Purpose	Define the sizes of various structures used by some conduit entry points.
Declared In	CondAPI.h
Constants	<pre>#define SZ_CFGCONDUITINFO sizeof(CFGCONDUITINFO) Defines the size of the CfgConduitInfoType structure.</pre> <pre>#define SZ_CONDUITREQUESTINFO sizeof(CONDUITREQUESTINFO) Defines the size of the ConduitRequestInfoType structure.</pre> <pre>#define SZ_REGISTRATIONINFO sizeof(REGISTRATIONINFO) Defines the size of the RegistrationInfoType structure.</pre>
Compatibility	HotSync Manager version: All. Exception: SZ_REGISTRATIONINFO is present only in versions 6.0 and later. Palm OS version: All.
See Also	CfgConduitInfoType , ConduitRequestInfoType , RegistrationInfoType

Structure Versions for Conduit Entry Points

Purpose Indicate the version numbers of certain data structures used by some conduit entry points.

Declared In CondAPI.h

Constants

```
#define CONDUITREQUESTINFO_VERSION_1 0x00000001
    This constant specifies version 1 of the
    ConduitRequestInfoType data structure.

#define CFGCONDUITINFO_VERSION_1 0x00000001
    This constant specifies version 1 of the
    CfgConduitInfoType data structure.
```

Compatibility HotSync Manager version: All.
Palm OS version: All.

See Also [ConduitRequestInfoType](#), [CfgConduitInfoType](#)

Conduit Entry Point API

Conduit-Defined Entry Point API Functions

Conduit-Defined Entry Point API Functions

This section describes the following entry points that a conduit implements.

<u>CfgConduit()</u>	Informs a conduit when the user selects it from HotSync Manager's Custom dialog box. Called only by HotSync Manager versions 3.0 and later (earlier versions call the <code>ConfigureConduit()</code> function instead).
<u>ConfigureConduit()</u>	Informs a conduit when the user selects it from HotSync Manager's Custom dialog box. HotSync Manager versions earlier than 3.0 call only this entry point, whereas versions 3.0 and later call <code>CfgConduit()</code> first and then call <code>ConfigureConduit()</code> only if the call to <code>CfgConduit()</code> is not successful.
<u>GetConduitInfo()</u>	(Required) Retrieves information about a conduit when called by HotSync Manager.
<u>GetConduitName()</u>	(Optional) Retrieves the display name of your conduit when called by HotSync Manager.
<u>GetConduitVersion()</u>	(Required) Retrieves the version number of your conduit when called by HotSync Manager.
<u>OpenConduit()</u>	(Required) Starts a conduit's synchronization process and provides a conduit information about the current HotSync session when this entry point is called by HotSync Manager.

CfgConduit Function

Purpose	Informs a conduit when the user selects it from HotSync Manager's Custom dialog box. Called only by HotSync Manager versions 3.0 and later (earlier versions call the ConfigureConduit() function instead).
Declared In	CondAPI.h
Prototype	<pre>SInt32 CfgConduit (ConduitCfgEnum <i>cfgType</i>, void *<i>pArgs</i>, UInt32 *<i>pdwArgssSize</i>) typedef SInt32 (*PCFGCONDUIT) (ConduitCfgEnum <i>cfgType</i>, void *<i>pArgs</i>, UInt32 *<i>pdwArgssSize</i>)</pre>
Parameters	<p>→ <i>cfgType</i> The version number of this entry point function. This is one of the ConduitCfgEnum values.</p> <p>↔ <i>pArgs</i> A pointer to a CfgConduitInfoType structure. HotSync Manager passes in the stored temporary and permanent synchronization preferences for the current HotSync user. Your conduit passes back updated synchronization preferences. This parameter cannot be NULL.</p> <p>→ <i>pdwArgssSize</i> The number of bytes in the structure referenced by the <i>pArgs</i> parameter.</p>
Returns	If successful, a conduit must return 0. If unsuccessful, a conduit must return a nonzero error code value. For more information about the error codes, see " Conduit Entry Point API Error Codes " on page 524.

Conduit Entry Point API

CfgConduit

Comments	<p>HotSync Manager calls a conduit's <code>CfgConduit()</code> entry point when a user decides to configure your conduit by clicking HotSync Manager > Custom. Usually a conduit responds by displaying a Change HotSync Action dialog box for the user to configure how the conduit performs during the next and all subsequent HotSync operations. For more background, see "User's Conduit Synchronization Preferences" on page 62 in the <i>Introduction to Conduit Development</i>.</p> <p>Use the information passed in the <code>pArgs</code> parameter to determine what your conduit should display as the user's previously stored preference. Modify the <code>syncNew</code> field with the user's new preference and the <code>syncPref</code> field to indicate whether it is permanent or temporary.</p> <p><code>CfgConduit()</code> is an updated version of ConfigureConduit(), which is used for the same purpose. This function receives different information (in the <code>pArgs</code> parameter) from what <code>ConfigureConduit()</code> does. Versions 3.0 and later of HotSync Manager attempt to call <code>CfgConduit()</code> before falling back to calling <code>ConfigureConduit()</code>. If your conduit must support HotSync Manager versions earlier than 3.0, then you must implement <code>ConfigureConduit()</code> also.</p> <p><code>PCFGCONDUIT</code> is defined as a pointer to the <code>CfgConduit()</code> entry point. This pointer is provided as a convenience for conduit developers who write tools to interrogate their own conduits.</p>
Compatibility	HotSync Manager version: 3.0 and later. Palm OS version: All.
See Also	ConfigureConduit() , CfgConduitInfoType

ConfigureConduit Function

Purpose Informs a conduit when the user selects it from HotSync Manager's **Custom** dialog box. HotSync Manager versions earlier than 3.0 call only this entry point, whereas versions 3.0 and later call [CfgConduit\(\)](#) first and then call ConfigureConduit() only if the call to CfgConduit() is not successful.

Declared In CondAPI.h

Prototype

```
SInt32 ConfigureConduit  
    (CSyncPreference &syncPrefs)  
typedef SInt32 (*PCONFIGURECONDUIT)  
    (CSyncPreference &syncPrefs)
```

Parameters

→ *syncPrefs*
An object of the [CSyncPreference](#) class, which contains information for your conduit.

Returns If successful, a conduit must return 0.

If unsuccessful, a conduit must return a nonzero error code value. For more information about the error codes, see "[Conduit Entry Point API Error Codes](#)" on page 524.

Comments As with [CfgConduit\(\)](#), HotSync Manager calls a conduit's ConfigureConduit() entry point when a user decides to configure your conduit by clicking **HotSync Manager > Custom**. Usually a conduit responds by displaying a **Change HotSync Action** dialog box for the user to configure how the conduit performs during the next and all subsequent HotSync operations. Use the information passed in the *syncPrefs* parameter to determine what your conduit should display.

However, ConfigureConduit() is an older version of [CfgConduit\(\)](#), which is used for the same purpose but provides different information. Versions of HotSync Manager earlier than 3.0 call ConfigureConduit() only; newer versions first attempt to call CfgConduit(), and fall back to calling this function if CfgConduit() is not available.

PCONFIGURECONDUIT is defined as a pointer to the ConfigureConduit() entry point. This pointer is provided as a convenience for conduit developers who write tools to interrogate their own conduits.

Conduit Entry Point API

ConfigureConduit

Compatibility HotSync Manager version: All (see comments).
Palm OS version: All.

See Also [CfgConduit\(\)](#)

GetConduitInfo Function

Purpose *(Required)* Retrieves information about a conduit when called by HotSync Manager.

Declared In CondAPI.h

Prototype

```
SInt32 GetConduitInfo (ConduitInfoEnum infoType,  
                      void *pInArgs, void *pOut, UInt32 *pdwOutSize)  
typedef SInt32 (*PGETCONDUITINFO)  
          (ConduitInfoEnum infoType, void *pInArgs,  
           void *pOut, UInt32 *pdwOutSize)
```

Parameters

→ *infoType*
The type of information that HotSync Manager is requesting.
This is one of the [ConduitInfoEnum](#) values.

→ *pInArgs*
If the value of *infoType* is eConduitName, this parameter
is a pointer to a [ConduitRequestInfoType](#) structure.
Otherwise, this value is NULL.

← *pOut*
A pointer to the information that HotSync Manager requests.
[Table 6.1](#) describes what this parameter should point to
depending on what HotSync Manager requests via the
infoType parameter.

← *pdwOutSize*
The size in bytes of the value that the *pOut* parameter points
to.

Returns If successful, a conduit must return 0.

If unsuccessful, a conduit must return a nonzero error code value.
For more information about the error codes, see “[Conduit Entry Point API Error Codes](#)” on page 524.

Conduit Entry Point API

GetConduitInfo

Comments HotSync Manager calls `GetConduitInfo()` to retrieve information about your conduit. Your implementation of this entry point must respond differently for each [ConduitInfoEnum](#) value passed in the *infoType* parameter. [Table 6.1](#) lists these values and how your conduit should respond.

Table 6.1 GetConduitInfo() requests and conduit responses

If the ConduitInfoEnum value passed in <i>infoType</i> is...	Then in <i>pOut</i> , pass back a pointer to...
eConduitName = 0	a TCHAR buffer.
eMfcVersion	a UInt32 value equal to one of the constants in “ MFC Versions ” on page 507.
eDefaultAction	an eSyncTypes enum value.
eRegistrationInfo	a RegistrationInfoType structure.
eDoNotDisplayInConduitListForUser	a value of 0, if your conduit should be displayed in the Custom dialog box. Return a nonzero value if it should not be displayed.
eRunAlways	a value of 0, if your conduit should be run only if a matching application is on the handheld. Return a nonzero value if your conduit should always be run.
eDoNotDisplayProgress	a value of 0, if your conduit should be displayed in the HotSync Progress dialog box. Return a nonzero value if it should not be displayed.

`PGETCONDUITINFO` is defined as a pointer to the `GetConduitInfo()` entry point. This pointer is provided as a convenience for conduit developers who write tools to interrogate their own conduits.

Compatibility

HotSync Manager version: All, except as noted below.
Palm OS version: All.

HotSync Manager versions 6.0 and later can pass in only these values in the *infoType* parameter:

- eConduitName
- eDefaultAction
- eRegistrationInfo
- eDoNotDisplayInCustomDialog
- eRunAlways
- eDoNotDisplayProgress

HotSync Manager versions *earlier than* 6.0 can pass in only these values in the *infoType* parameter:

- eConduitName
- eMfcVersion
- eDefaultAction

IMPORTANT: If your conduit must work with HotSync Manager versions earlier than 6.0, your implementation of `GetConduitInfo()` must return the appropriate MFC version constant. If you are recompiling a conduit you created with an older version of the CDK and did not originally implement `GetConduitInfo()` (now required), HotSync Manager assumes that your conduit is built on MFC 4.1. If it is not, HotSync Manager crashes when it calls your conduit.

HotSync Manager versions 6.0 and later do not need to check a conduit's MFC version, so they never pass in the `eMfcVersion` value via the *infoType* parameter.

See Also

[ConduitInfoEnum](#), [ConduitRequestInfoType](#)

Conduit Entry Point API

GetConduitName

GetConduitName Function

Purpose	(Optional) Retrieves the display name of your conduit when called by HotSync Manager.
Declared In	CondAPI.h
Prototype	<pre>SInt32 GetConduitName (char *name, UInt16 nLen) typedef SInt32 (*PGETCONDUITNAME) (char *name, (UInt16 nLen)</pre>
Parameters	<p>← <i>name</i> The name of your conduit.</p> <p>→ <i>nLen</i> The maximum number of bytes that your conduit can store in the character buffer that the <i>name</i> parameter points to.</p>
Returns	If successful, a conduit must return 0. If unsuccessful, a conduit must return a nonzero error code value. For more information about the error codes, see “ Conduit Entry Point API Error Codes ” on page 524.
Comments	<p>HotSync Manager versions 3.0.1 and later execute the following sequence of actions until one of them succeeds in retrieving the name of your conduit:</p> <ol style="list-style-type: none">1. Read the Name entry written with the CondCfg utility or the Conduit Manager API during conduit installation.2. Else call GetConduitInfo().3. Else call <code>GetConduitName()</code>. <p>Therefore you are not required to implement <code>GetConduitName()</code> if your conduit needs to be compatible with only HotSync Manager versions 3.0.1 and later.</p> <p>HotSync Manager versions 3.0 and earlier require that your conduit implement <code>GetConduitName()</code>. In which case, your implementation must fill in the name buffer with a string that you want HotSync Manager to display to the user.</p> <p><code>PGETCONDUITNAME</code> is defined as a pointer to the <code>GetConduitName()</code> entry point. This pointer is provided as a convenience for conduit developers who write tools to interrogate their own conduits.</p>

Compatibility HotSync Manager version: Optional for 3.0.1 and later, required for 3.0 and earlier. See “Comments” for version-specific differences.
Palm OS version: All.

See Also [GetConduitInfo\(\)](#)

Conduit Entry Point API

GetConduitVersion

GetConduitVersion Function

Purpose	(Required) Retrieves the version number of your conduit when called by HotSync Manager.
Declared In	CondAPI.h
Prototype	<code>UInt32 GetConduitVersion ()</code> <code>typedef UInt32 (*PGETCONDUITVERSION) ()</code>
Parameters	None.
Returns	The version number of your conduit, packed into the <code>UInt32</code> return value as follows: <ul style="list-style-type: none">• Major version number is <code>HIBYTE(LOWORD)</code>• Minor version number is <code>LOBYTE(LOWORD)</code>
Comments	HotSync Manager calls <code>GetConduitVersion()</code> to retrieve the version of your conduit that is running on the desktop computer. Your implementation must pack your major version number into the high byte of the low word and your minor version number into the low byte of the low word in the result. <code>PGETCONDUITVERSION</code> is defined as a pointer to the <code>GetConduitVersion()</code> entry point. This pointer is provided as a convenience for conduit developers who write tools to interrogate their own conduits.
Compatibility	HotSync Manager version: All. Palm OS version: All.

OpenConduit Function

Purpose (*Required*) Starts a conduit's synchronization process and provides a conduit information about the current HotSync session when this entry point is called by HotSync Manager.

Declared In CondAPI.h

Prototype

```
SInt32 OpenConduit (PROGRESSFN pFn,
                     CSyncProperties &syncProps)
typedef SInt32 (*POPCONDUIT) (PROGRESSFN pFn,
                             CSyncProperties &syncProps)
```

Parameters

→ *pFn*
A pointer to a callback function that HotSync Manager passes to your conduit. For more information, see [PROGRESSFN\(\)](#).

→ *syncProps*
Properties of the current HotSync operation, including the type of synchronization, current user name, and so on. This information is defined in an object of the [CSyncProperties](#) class.

Returns If successful, a conduit must return 0.

If unsuccessful, a conduit must return a nonzero error code value. For more information about the error codes, see “[Conduit Entry Point API Error Codes](#)” on page 524.

Comments HotSync Manager calls a conduit's `OpenConduit()` entry point to begin the process of synchronizing data between the desktop computer and the handheld. This is where the main work of a conduit is done. In the *syncProps* parameter, HotSync Manager passes in information about the current HotSync session that a conduit uses to determine what it needs to do during the current HotSync session.

The `syncProps.m_SyncType` member tells a conduit the type of synchronization operation to perform based on the user's saved preference (synchronize, handheld overwrites desktop, desktop overwrites handheld, do nothing), and if “synchronize” then whether the conduit should perform a fast or slow sync.

Conduit Entry Point API

OpenConduit

IMPORTANT: The `eFast` or `eSlow` value passed to a conduit in the `m_SyncType` field is based only on whether the last HotSync operation was with the current desktop. For non-schema databases, this is sufficient for a conduit to determine whether to perform a fast or slow sync. However, for schema databases, this value is not sufficient. Instead, call `SyncDbGetSyncMode()` to take full advantage of the extra change tracking information that is available only in schema databases and not in classic and extended databases.

For conduits using the Generic Conduit Framework, `OpenConduit()` does the following:

- instantiate an object of the `CSynchronizer` class and specify whether your conduit needs to handle categories or the application info block
- engage your synchronization by calling the `CSynchronizer`'s `Perform()` method
- delete the conduit's `CSynchronizer` object

`POPCONDUIT` is defined as a pointer to the `OpenConduit()` entry point. This pointer is provided as a convenience for conduit developers who write tools to interrogate their own conduits.

Compatibility HotSync Manager version: All.
Palm OS version: All.

See Also [CSyncProperties](#), [PROGRESSFN\(\)](#), [SyncDbGetSyncMode\(\)](#)

HotSync Manager Callback Function

This section defines the [PROGRESSFN\(\)](#) function that a conduit can use to call back into HotSync Manager from its [OpenConduit\(\)](#) entry point.

PROGRESSFN Function

Purpose	Displays a message in the HotSync Progress dialog while a conduit is running when called by a conduit.
Declared In	CondAPI.h
Prototype	<code>typedef SInt32 (*PROGRESSFN) (char *progress)</code>
Parameters	<code>→ progress</code> A character string to be displayed in the Status field of the HotSync Progress dialog box.
Returns	Always returns 0.
Comments	PROGRESSFN() is a callback function. HotSync Manager passes your conduit a pointer to this function when it calls your OpenConduit() entry point. Your conduit can call this function to make HotSync Manager display a message in the Status field of the HotSync Progress dialog box while your conduit is running.
Compatibility	HotSync Manager version: All. Palm OS version: All.
See Also	OpenConduit()

Conduit Entry Point API Error Codes

[Table 6.2](#) lists the values of error codes that your conduit's entry points can return to HotSync Manager. Returning one of these errors causes HotSync Manager to write a similar error message to the HotSync log. However, PalmSource recommends that you return CONDERR_NONE if successful and -1 if unsuccessful and call the HotSync Log API to add your own more specific messages to the log.

All of the named error codes below are defined as preprocessor constants, which are declared in the CondAPI.h header file.

Table 6.2 Conduit Entry Point API error codes

Value	Code	Description
-1	—	Any value other than those listed below indicates an unspecified error.
0x0000	CONDERR_NONE	The function call was successful.
0x1000	CONDERR_FIRST	No entry point should return this value. It is only the offset from which all other error codes are based.
0x1001	CONDERR_NO_REMOTE_CATEGORIES	There are no categories defined on the handheld.
0x1002	CONDERR_NO_LOCAL_CATEGORIES	There are no categories defined on the desktop computer.
0x1003	CONDERR_SAVE_REMOTE_CATEGORIES	The categories could not be saved on the handheld.

Table 6.2 Conduit Entry Point API error codes (*continued*)

Value	Code	Description
0x1004	CONDERR_BAD_REMOTE_TABLES	This error code is deprecated in HotSync Manager 6.0 and later.
0x1005	CONDERR_BAD_LOCAL_TABLES	This error code is deprecated in HotSync Manager 6.0 and later.
0x1006	CONDERR_BAD_LOCAL_BACKUP	This error code is deprecated in HotSync Manager 6.0 and later.
0x1007	CONDERR_ADD_LOCAL_RECORD	This error code is deprecated in HotSync Manager 6.0 and later.
0x1008	CONDERR_ADD_REMOTE_RECORD	Could not add a record on the handheld.
0x1009	CONDERR_CHANGE_REMOTE_RECORD	Could not change a record on the handheld.
0x100A	CONDERR_RAW_RECORD_ALLOCATE	Could not allocate memory for a record.
0x100B	CONDERR_REMOTE_CHANGES_NOT_SENT	The changes were not successfully sent to the handheld.
0x100C	CONDERR_LOCAL_MEMORY_ALLOC_FAILED	Could not allocate memory on the desktop computer.
0x100D	CONDERR_CONVERT_TO_REMOTE_CATS	Could not convert a record's category to a category on the handheld.

Conduit Entry Point API

Conduit Entry Point API Error Codes

Table 6.2 Conduit Entry Point API error codes (*continued*)

Value	Code	Description
0x100E	CONDERR_CONVERT_TO_LOCAL_CATS	Could not convert a record's category to a category on the desktop computer.
0x100F	CONDERR_CONVERT_TO_REMOTE_REC	Could not convert a desktop computer record to a record on the handheld.
0x1010	CONDERR_CONVERT_FROM_REMOTE_REC	Could not convert a handheld record to a record on the desktop computer.
0x1011	CONDERR_REMOTE_RECS_NOT_PURGED	The records on the handheld could not be successfully purged.
0x1012	CONDERR_BAD_SYNC_TYPE	The specified synchronization type is not valid.
0x1013	CONDERR_ABORT_DB_INSTALL	Abort database install, for example, when the handheld runs out of memory.
0x1050	CONDERR_DATE_MOVED	Dates before 1970 were moved to a valid date.
0x1060	CONDERR_SUBSCRIBE_FAILED	The file link failed. This error code is deprecated because the file link feature has been removed in HotSync Manager 6.0.1 and later.

Table 6.2 Conduit Entry Point API error codes (*continued*)

Value	Code	Description
0x1070	CONDERR_UNSUPPORTED_CONDUITINFO_ENUM	The specified ConduitInfoEnum value is not supported.
0x1071	CONDERR_INVALID_PTR	A pointer is not valid.
0x1072	CONDERR_BUFFER_TOO_SMALL	The buffer is too small to contain the data.
0x1073	CONDERR_INVALID_BUFFER_SIZE	The buffer size is not valid for the buffer.
0x1074	CONDERR_INVALID_INARGS_PTR	The input arguments pointer is not valid.
0x1075	CONDERR_INVALID_INARGS_STRUCT	The input arguments structure is not valid.
0x1076	CONDERR_CONDUIT_RESOURCE_FAILURE	Could not read a resource from your conduit's DLL.
0x1077	CONDERR_INVALID_OUTSIZE_PTR	The output size pointer is not valid.
0x1078	CONDERR_INVALID_ARGSSIZE_PTR	The output arguments size is not valid.
0x1079	CONDERR_UNSUPPORTED_CFGCONDUIT_ENUM	The specified ConduitCfgEnum value is not supported.
0x107A	CONDERR_INVALID_ARGSSIZE	The argument size is not valid.
0x107B	CONDERR_UNSUPPORTED_STRUCT_VERSION	The version number of the structure is not supported on the handheld.
0x107C	CONDERR_NOCLIENTINFO_AVAILABLE	No client information available.

Conduit Entry Point API

Conduit Entry Point API Error Codes

HotSync Log API

The HotSync® Log API enables conduits to queue messages that are later written to the HotSync log. The messages are written to the HotSync log just before the synchronization process ends.

The HotSync Log functions are available in `HSLog20.dll` and declared in `HSLog.h`.

The sections in this chapter are:

HotSync Log Constants	529
HotSync Log Functions	534
HotSync Log API Error Codes	538

For more information on the HotSync log, see “[Adding Messages to the HotSync Log](#)” on page 46 in the *Introduction to Conduit Development*.

HotSync Log Constants

This section describes the enumerated constant [Activity](#) that you can use with HotSync log functions.

Activity Enum

Purpose	Specifies the activity that causes a conduit to call <code>LogAddEntry()</code> or <code>LogAddFormattedEntry()</code> to add an entry to the log.
Declared In	<code>HSLog.h</code>
Constants	<code>s1Text = -1</code> Allows a conduit to add text to the log without incrementing the warning counter. <code>s1DoubleModify</code> A record has been modified on both the desktop computer and handheld. <code>s1DoubleModifyArchive</code> A record that has been modified on both the desktop and handheld has been archived. <code>s1ReverseDelete</code> A record that was deleted on one side has been restored, because the same record was modified on the other side. <code>s1TooManyCategories</code> No more categories can be added. <code>s1CategoryDeleted</code> A category was deleted. <code>s1DateChanged</code> The date was changed. <code>s1CustomLabel</code> A custom label was changed. <code>s1ChangeCatFailed</code> Changing a category failed. <code>s1RemoteReadFailed</code> Reading a record failed on the handheld. <code>s1RemoteAddFailed</code> Adding a record on the handheld failed. <code>s1RemotePurgeFailed</code> Purging a record on the handheld failed. <code>s1RemoteChangeFailed</code> Changing a record on the handheld failed.

slRemoteDeleteFailed
Deleting a record on the handheld failed.

slLocalAddFailed
Adding a record on the desktop computer failed.

slRecCountMismatch
Record counts did not match.

slXMapFailed
The position cross-map operation failed.

slArchiveFailed
The archive operation failed.

slLocalSaveFailed
Saving data on the desktop computer failed.

slResetFlagsFailed
Resetting the synchronization flags failed.

slSyncStarted
A conduit started its synchronization operations.

slSyncFinished
A conduit completed its synchronization operations successfully.

slSyncAborted
The synchronization operation was aborted.

slWarning
This constant lets a conduit record a warning that doesn't fit any of the other activity codes provided.

slDoubleModifySubsc
A file link record was modified on the desktop. This value is deprecated because the file link feature has been removed in HotSync Manager 6.0.1 and later.

slFileLinkCompleted
Processing of a file link completed. This value is deprecated because the file link feature has been removed in HotSync Manager 6.0.1 and later.

slFileLinkDeleted
A file link was deleted. This value is deprecated because the file link feature has been removed in HotSync Manager 6.0.1 and later.

HotSync Log API

Activity

s1SyncDidNothing

The user specified that the conduit should not perform any operations during synchronization. You can use this value only with HotSync Manager version 6.0 or later.

s1SyncSessionStart

The HotSync operation started. Only HotSync Manager version 6.0.1 or later uses this value; conduits must not.

s1SyncSessionEnd

The HotSync operation completed. Only HotSync Manager version 6.0.1 or later uses this value; conduits must not.

s1SyncSessionCancelled

The user clicked **Cancel** on the **HotSync Progress** dialog box. Only HotSync Manager version 6.0.1 or later uses this value; conduits must not.

s1Error

An error occurred. You can use this value only with HotSync Manager version 6.0.1 or later.

s1Recommendation

Recommendation that the user do something—for example, resolve conflicts that a conduit could not. You can use this value only with HotSync Manager version 6.0.1 or later.

s1HTMLText

This type of log entry contains HTML tags or characters. Using this value passes the string to the log unchanged so that its HTML formatting will be rendered. All other Activity values cause the HotSync Log API to replace HTML control characters (<, >, &) with their HTML equivalents (<, >, &). You can use this value only with HotSync Manager version 6.0.1 or later.

Comments

Your conduit passes one of these values to [LogAddEntry\(\)](#) or [LogAddFormattedEntry\(\)](#) to indicate what type of entry it is adding to the log.

For more information, see “[Adding Messages to the HotSync Log](#)” on page 46 in the *Introduction to Conduit Development*.

Compatibility HotSync Manager version: All, with exceptions described for each value above.
Palm OS version: All.

See Also [LogAddEntry\(\)](#), [LogAddFormattedEntry\(\)](#)

HotSync Log Functions

This section describes the following functions, which allow you to write to the HotSync log.

<u>LogAddEntry()</u>	Adds a message to the HotSync log on the desktop.
<u>LogAddFormattedEntry()</u>	Adds a formatted message to the HotSync log on the desktop.
<u>LogTestCounters()</u>	Determines the number of messages that have been added to the HotSync log on the desktop.

LogAddEntry Function

Purpose	Adds a message to the HotSync log on the desktop.
Declared In	HSLog.h
Prototype	<code>long LogAddEntry (LPCTSTR pszEntry, Activity act, BOOL bTimeStamp)</code>
Parameters	<p>→ <i>pszEntry</i> The string to enter into the log.</p> <p>→ <i>act</i> The activity that causes a conduit to add an entry to the log. This is one of the Activity enum values.</p> <p>→ <i>bTimeStamp</i> If true, HotSync Manager includes a time-stamp with the log entry.</p>
Returns	If successful, returns <code>s1NoError</code> . If unsuccessful, returns -1.
Comments	For more information on the HotSync log, see “ Adding Messages to the HotSync Log ” on page 46 in the <i>Introduction to Conduit Development</i> .
Compatibility	HotSync Manager version: All. Palm OS version: All.
See Also	LogAddFormattedEntry() , Activity

HotSync Log API

LogAddFormattedEntry

LogAddFormattedEntry Function

Purpose Adds a formatted message to the HotSync log on the desktop.

Declared In HSLog.h

Prototype long LogAddFormattedEntry (Activity *act*,
 BOOL *bTimeStamp*, const char **dataString*, ...)

Parameters

→ *act*

The activity that causes a conduit to add an entry to the log.
This is one of the [Activity](#) enum values.

→ *bTimeStamp*

If true, HotSync Manager includes a time-stamp with the log entry.

→ *dataString*

The string containing formatting placeholders. This string can contain the same formatting instructions as do strings that are passed to standard C Library functions such as `sprintf()`.

→ ...

Parameter values for the formatting placeholders in the *dataString* parameter.

Returns If successful, returns `s1NoError`.

If unsuccessful, returns -1.

Comments For more information on the HotSync log, see “[Adding Messages to the HotSync Log](#)” on page 46 in the *Introduction to Conduit Development*.

IMPORTANT: The *dataString* parameter must point to a string that is 256 characters or shorter. Longer strings can cause HotSync Manager to crash. If you must add formatted strings longer than 256 characters to the log, you can use `sprintf()` with your own buffer and then call `LogAddEntry()` to print your buffer to the log.

Compatibility HotSync Manager version: All.
Palm OS version: All.

See Also [LogAddEntry\(\)](#), [Activity](#)

LogTestCounters Function

Purpose	Determines the number of messages that have been added to the HotSync log on the desktop.
Declared In	HSLog.h
Prototype	WORD LogTestCounters ()
Parameters	None.
Returns	Returns the number of messages in the log. Returns 0 if the log does not contain any messages.
Comments	The LogTestCounters () function does <i>not</i> count log entries for the following activity types: <ul style="list-style-type: none">• slSyncAborted• slSyncFinished• slSyncStarted• slText The LogTestCounters () function does count log entries for any other activity types. Note that HotSync Manager displays a message upon completion noting that messages were logged; this message is displayed only when one or more of the log entries counted by LogTestCounters () is written to the log. Most conduits do not need to use this function. It is usually used only by HotSync Manager.
Compatibility	HotSync Manager version: All. Palm OS version: All.

HotSync Log API

HotSync Log API Error Codes

HotSync Log API Error Codes

[Table 7.1](#) lists the values of error codes that some HotSync Log API functions can return. The description of each function states which errors each function can return. Note that only the values -1 and `s1NoError` are returned by any of the public functions of this API. The others are used only by private functions.

All of the named error codes below are defined as values of the `.LogError` enum, which is declared in the `HSLog.h` header file.

Table 7.1 HotSync Log API error codes

Value	Code	Description
-1	—	The function was unsuccessful.
0	<code>s1NoError</code>	No error.
1	<code>s1BadStream</code>	The stream is invalid.
2	<code>s1DeleteFileFailed</code>	The old log file cannot be deleted.
3	<code>s1MoveFileFailed</code>	The working log file could not be moved to the specified.

Palm OS Common Language API

The Palm OS® Common Language API enables your conduit to load and unload language resource DLLs. For example, if a conduit requires a French version of the resources, it can call these functions to display the French resources from a DLL containing only French resources. If the conduit cannot find the French DLL, it defaults to the base language of the conduit DLL's resources.

IMPORTANT: If your conduit uses this method of selecting language resources, create a unique name for your language resource DLL.

The Palm OS Common Language functions are available in `PalmCmn.dll` and declared in `PALM_CMN.H` and `LANG_DLL.h`.

The sections in this chapter are:

Palm OS Common Language Constants	540
Palm OS Common Language Functions	541

Palm OS Common Language Constants

Purpose Defines the language of the resource DLL.

Declared In LANG_DLL.h

Constants

```
#define LANGUAGE_DUTCH 0x0200
    Dutch language resources.

#define LANGUAGE_ENGLISH 0x0010
    English language resources.

#define LANGUAGE_FRENCH 0x0020
    French language resources.

#define LANGUAGE_GERMAN 0x0040
    German language resources.

#define LANGUAGE_ITALIAN 0x0100
    Italian language resources.

#define LANGUAGE_JAPANESE 0x1000
    Japanese language resources.

#define LANGUAGE_PORTUGUESE 0x0400
    Portuguese language resources.

#define LANGUAGE_SCHINESE 0x2000
    Simplified Chinese language resources.

#define LANGUAGE_SPANISH 0x0080
    Spanish language resources.

#define LANGUAGE_TCHINESE 0x4000
    Traditional Chinese language resources.
```

Comments [PalmGetResourceVersion\(\)](#) returns one of these to indicate the language resource of the specified DLL.

Compatibility Palm OS Common Language API version: 1
Palm OS version: All.

Palm OS Common Language Functions

This section describes the following Palm OS Common Language functions.

<u>PalmFreeLanguage()</u>	Unloads the specified language resource DLL.
<u>PalmGetResourceVersion()</u>	Returns the language resource of the specified DLL.
<u>PalmGetVersion()</u>	Returns the version of the Palm OS Common Language API.
<u>PalmLoadLanguage()</u>	Loads the specified language resource DLL.

PalmFreeLanguage Function

Purpose	Unloads the specified language resource DLL.
Declared In	PALM_CMN.H
Prototype	BOOL PalmFreeLanguage (HINSTANCE <i>hRscInst</i> , HINSTANCE <i>hAppInst</i>)
Parameters	<p>→ <i>hRscInst</i> The instance handle of the language resource to unload. Set this value to the return value of PalmLoadLanguage().</p> <p>→ <i>hAppInst</i> The instance handle of the application calling this function.</p>
Returns	If successful, returns true. If unsuccessful, returns false.
Comments	To free the language resource DLL that was loaded by PalmLoadLanguage, pass the PalmFreeLanguage function the value that PalmLoadLanguage returned, which is the handle of the language resource DLL it loaded. If the <i>hRscInst</i> and <i>hAppInst</i> handles are the same, this function does nothing (because there is nothing to free) and returns true.
Compatibility	Palm OS Common Language API version: 1 Palm OS version: All.

PalmGetResourceVersion Function

Purpose	Returns the language resource of the specified DLL.
Declared In	PALM_CMN.H
Prototype	DWORD PalmGetResourceVersion (HINSTANCE <i>hLangInstance</i>)
Parameters	→ <i>hLangInstance</i> The instance handle of the language resource DLL
Returns	Returns one of the constants that specifies the language of the specified DLL (see “ Palm OS Common Language Constants ” on page 540).
Compatibility	Palm OS Common Language API version: 1 Palm OS version: All.

PalmGetVersion Function

Purpose	Returns the version of the Palm OS Common Language API.
Declared In	PALM_CMN.H
Prototype	DWORD PalmGetVersion (void)
Parameters	None.
Returns	Returns the version number of the Palm OS Common Language API.
Compatibility	Palm OS Common Language API version: 1 Palm OS version: All.

PalmLoadLanguage Function

Purpose	Loads the specified language resource DLL.
Declared In	PALM_CMN.H
Prototype	HINSTANCE PalmLoadLanguage (LPCTSTR <i>pFileName</i> , HINSTANCE <i>hAppInst</i> , DWORD * <i>pdwVersion</i>)
Parameters	<p>→ <i>pFileName</i> A pointer to the filename of the DLL to load.</p> <p>→ <i>hAppInst</i> The instance handle of the application calling this function.</p> <p>← <i>pdwVersion</i> A pointer to one of the constants that specifies the language of the DLL that the function loaded (see “Palm OS Common Language Constants” on page 540).</p>
Returns	If this function finds the specified DLL, it returns an instance handle to the language resource DLL it loaded. If it does not find the specified DLL, it returns the instance handle passed to it in <i>hAppInst</i> —that is, the handle of whatever called this function.
Comments	The return value is the handle of the language resource DLL that the function loaded. For example, if your conduit includes the call
	<hr/> <pre>hRscInstance = PalmLoadLanguage("MYCLANG.DLL" , hDLLInstance, &dwVersion);</pre> <hr/>
	then PalmLoadLanguage looks for (and attempts to load) MYCLANG.DLL, which contains the localized resources for your conduit. If it successfully loads it, this function returns the instance handle. If it cannot find the DLL or loads unsuccessfully for any other reason, PalmLoadLanguage returns the instance handle of your conduit DLL, and therefore your conduit uses the resources compiled into the conduit DLL itself.
	In the PalmFreeLanguage() function, set <i>hRscInst</i> to this value.
Compatibility	Palm OS Common Language API version: 1 Palm OS version: All.

Palm OS Common Language API

PalmLoadLanguage

Expansion Manager API

The Expansion Manager on the handheld is an optional system extension that adds support for hardware expansion cards on Palm Powered™ handhelds. The handheld Expansion Manager's primary function is to manage slots on the handheld and the drivers associated with those slots. Individual slot drivers on the handheld—which are provided by handheld manufacturers—provide support for various expansion card types including Secure Digital (SD), MultiMediaCard (MMC), CompactFlash, Sony's Memory Stick, and others.

The API documented in this chapter provides conduits an interface to the Expansion Manager on the handheld during a HotSync® operation. Through this interface, conduits can determine whether an expansion card is present in a slot and get information about that card.

This chapter provides the following information about the Expansion Manager API:

Expansion Manager Structures	549
Expansion Manager Constants	550
Expansion Manager Functions	554
Expansion Manager API Error Codes	560

The desktop Expansion Manager functions are available in `VFSAPI.dll` and declared in `ExpansionMgr.h`. (Expansion Manager error codes are declared in `VFSERR.h`.) For more information on the Expansion Manager, see [Chapter 6, “Using Expansion Technology,”](#) on page 49 of the *C/C++ Sync Suite Companion*.

NOTE: The Expansion Manager is an optional system extension on handhelds. Therefore you should check for the presence of the Expansion Manager on the handheld before calling any Expansion Manager API functions. See “[Verifying Handheld Compatibility](#)” on page 60 in the *C/C++ Sync Suite Companion..*

Expansion Manager Structures

This section describes the [ExpCardInfoType](#) structure defined for the Expansion Manager.

ExpCardInfoType Struct

Purpose Receives the characteristics of the card loaded in the expansion slot.

Declared In ExpansionMgr.h

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. This is set to one or more of the [Hardware Capability Flags](#).

manufacturerStr
Names the manufacturer of the card—for example, “Palm” or “Motorola”.

productStr
Name of the product—for example, “SafeBackup 32 MB”.

deviceClassStr
Describes the type of card—for example, “Backup” or “Ethernet”.

deviceUniqueIDStr
Unique identifier for the product—for example, a serial number. This value is set to the empty string if no identifier exists.

Comments This structure is initialized by the underlying slot driver.

Expansion Manager API

Expansion Manager Constants

Expansion Manager Constants

This section describes the following groups of preprocessor constants that you use the Expansion Manager API.

Directory Enumeration Constants	Control the process of directory enumeration when using VFSDirEntryEnumerate() .
Hardware Capability Flags	Indicates the capabilities of the card in ExpCardInfoType .capabilityFlags.
Media Type Constants	Define the media types supported by the Expansion Manager.
Miscellaneous Expansion Manager Constants	Define miscellaneous constants used by Expansion Manager.

Directory Enumeration Constants

Purpose	Control the process of directory enumeration when using VFSDirEntryEnumerate() .
Declared In	<code>ExpansionMgr.h</code>
Constants	<pre>#define expIteratorStart 0L Supply this value to VFSDirEntryEnumerate() to begin enumeration. #define expIteratorStop 0xffffffffL When enumeration reaches the last directory entry, VFSDirEntryEnumerate() sets its directory entry iterator parameter to this value.</pre>
Compatibility	Expansion Manager version: All. Palm OS version: 4.0 or later. See “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i> for ways to confirm the presence of the Expansion Manager on the handheld.
See Also	VFSDirEntryEnumerate()

Hardware Capability Flags

Purpose	Indicates the capabilities of the card in ExpCardInfoType .capabilityFlags.
Declared In	<code>ExpansionMgr.h</code>
Constants	<pre>#define expCapabilityHasStorage 0x00000001 Indicates that the card has data storage. The expCapabilityReadOnly flag indicates whether the card can be written or only read, though. #define expCapabilityReadOnly 0x00000002 Indicates that the card is read-only.</pre>
Compatibility	Expansion Manager version: All. Palm OS version: 4.0 or later. See “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i> for ways to confirm the presence of the Expansion Manager on the handheld.
See Also	ExpCardInfoType

Expansion Manager API

Media Type Constants

Media Type Constants

Purpose	Define the media types supported by the Expansion Manager.
Declared In	<code>ExpansionMgr.h</code>
Constants	<pre>#define ExpMediaType_Any 'wild' Matches all media types when looking up a default directory. #define ExpMediaType_CompactFlash 'cfsh' CompactFlash. #define ExpMediaType_MemoryStick 'mstk' Memory Stick. #define ExpMediaType_MultiMediaCard 'mmcd' MultiMediaCard. #define ExpMediaType_PlugNPlay 'pnps' Universal “plug and play” (PnP) connector. #define ExpMediaType_PoserHost 'pose' Host file system emulated by Palm OS® Emulator. #define ExpMediaType_RAMDisk 'ramd' A RAM-disk-based media. #define ExpMediaType_SecureDigital 'sdig' Secure Digital. #define ExpMediaType_SmartMedia 'smed' SmartMedia.</pre>
Comments	These media types are used with ExpSlotMediaType() and VFSVolumeInfo() in the <code>VolumeInfoType.mediaType</code> field.
Compatibility	Expansion Manager version: All. Palm OS version: 4.0 or later. See “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i> for ways to confirm the presence of the Expansion Manager on the handheld.
See Also	ExpSlotMediaType() , VFSVolumeInfo()

Miscellaneous Expansion Manager Constants

Purpose	Define miscellaneous constants used by Expansion Manager.
Declared In	<code>ExpansionMgr.h</code>
Constants	<pre>#define expCardInfoStringMaxLen 31 Defines the maximum length of a string in a member of the ExpCardInfoType structure. #define expFtrIDVersion 0 To obtain the version of the Expansion Manager, you can call SyncReadFeature(), supplying a feature creator of sysFileCExpansionMgr and this constant for the feature number. However, the primary method for getting the version number is to call VFSsupport(). #define sysFileCExpansionMgr 'expn' Defines the feature creator ID for the Expansion Manager. See the description of expFtrIDVersion.</pre>
Compatibility	<p>Expansion Manager version: All. Palm OS version: 4.0 or later.</p> <p>See “Checking for Expansion Cards” on page 59 in the <i>C/C++ Sync Suite Companion</i> for ways to confirm the presence of the Expansion Manager on the handheld.</p>

Expansion Manager API

Expansion Manager Functions

Expansion Manager Functions

This section describes the following functions, which enable conduits to determine whether an expansion card is present in a slot and to get information about that card.

<u>ExpCardInfo()</u>	Retrieves information about an expansion card in a given slot.
<u>ExpCardPresent()</u>	Determines whether a card is present in the given slot.
<u>ExpSlotEnumerate()</u>	Enumerates the valid slots to obtain a list of slot reference numbers.
<u>ExpSlotMediaType()</u>	Obtains the media type identifier for the specified slot.

ExpCardInfo Function

Purpose	Retrieves information about an expansion card in a given slot.
Declared In	<code>ExpansionMgr.h</code>
Prototype	<code>SInt32 ExpCardInfo (UInt16 slotRefNumber, ExpCardInfoType *pCardInfo, void *pVoid)</code>
Parameters	<p>→ <i>slotRefNumber</i> The slot reference number passed back by ExpSlotEnumerate().</p> <p>← <i>pCardInfo</i> Pointer to ExpCardInfoType structure that contains information about the card in the specified slot.</p> <p>↔ <i>pVoid</i> This parameter is unused in this version of the Expansion Manager. Pass in NULL and ignore the value passed back.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error codes:</p> <p><code>expErrCardNoSectorReadWrite</code> <code>expErrCardNotPresent</code> <code>expErrInvalidSlotRefNumber</code> <code>expErrSlotDeallocated</code> <code>expErrUnsupportedOperation</code></p> <p>For descriptions of all error codes, see “Expansion Manager API Error Codes” on page 560.</p>
Comments	This routine returns information about a card, including whether the card supports secondary storage or is strictly read-only, by filling in the <code>ExpCardInfoType</code> structure’s fields.
Compatibility	<p>Expansion Manager version: All.</p> <p>Palm OS version: 4.0 or later.</p> <p>See “Checking for Expansion Cards” on page 59 in the <i>C/C++ Sync Suite Companion</i> for ways to confirm the presence of the Expansion Manager on the handheld.</p>
See Also	ExpCardPresent() , ExpSlotEnumerate()

Expansion Manager API

ExpCardPresent

ExpCardPresent Function

Purpose	Determines whether a card is present in the given slot.
Declared In	<code>ExpansionMgr.h</code>
Prototype	<code>SInt32 ExpCardPresent (UInt16 slotRefNumber, void *pVoid)</code>
Parameters	<p>→ <i>slotRefNumber</i> The slot reference number passed back by ExpSlotEnumerate().</p> <p>↔ <i>pVoid</i> This parameter is unused in this version of the Expansion Manager. Pass in NULL and ignore the value passed back.</p>
Returns	If successful, returns SYNCERR_NONE. If unsuccessful, returns one of the following error codes: <code>expErrCardNotPresent</code> <code>expErrInvalidSlotRefNumber</code> <code>expErrSlotDeallocated</code> <code>expErrUnsupportedOperation</code> For descriptions of all error codes, see “ Expansion Manager API Error Codes ” on page 560.
Comments	Call this function to test whether a card is present in a slot before making any VFS Manager API calls to access files on a card.
Compatibility	Expansion Manager version: All. Palm OS version: 4.0 or later. See “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i> for ways to confirm the presence of the Expansion Manager on the handheld.
See Also	ExpCardInfo() , ExpSlotEnumerate()

ExpSlotEnumerate Function

Purpose	Enumerates the valid slots to obtain a list of slot reference numbers.
Declared In	ExpansionMgr.h
Prototype	<pre>SInt32 ExpSlotEnumerate (UInt16 *pNumSlotRefListEntires, UInt16 *pSlotRefNumList, void *pVoid)</pre>
Parameters	<p>↔ <i>pNumSlotRefListEntires</i> On entry, a pointer to the number of slot reference numbers allocated. On return, it is a pointer to the number of slot reference numbers filled into <i>pSlotRefNumList</i>.</p> <p>← <i>pSlotRefNumList</i> A pointer to an array of slot reference numbers. The caller must allocate this buffer before calling this function.</p> <p>↔ <i>pVoid</i> This parameter is unused in this version of the Expansion Manager. Pass in NULL and ignore the value passed back.</p>
Returns	<p>If successful, returns SYNCERR_NONE.</p> <p>If unsuccessful, returns one of the following error codes: <i>expErrUnsupportedOperation</i> For descriptions of all error codes, see “Expansion Manager API Error Codes” on page 560.</p>
Comments	This function passes back a list of slot reference numbers for slots on the handheld. Note that you must allocate sufficient space for <i>pSlotRefNumList</i> before calling this function.
Example	The following example shows a way to allocate sufficient space before calling <code>ExpSlotEnumerate()</code> .

```
WORD wSlotRefList[32]; // Buffer for slot reference numbers.
WORD wSlotRefCount;    // Number of entries allocated for list.
long retval;

// Allocate enough space for buffer.
wSlotRefCount = sizeof (wSlotRefList) / sizeof (wSlotRefList[0]);
retval = ExpSlotEnumerate(&wSlotRefCount, wSlotRefList, NULL);
```

Expansion Manager API

ExpSlotEnumerate

Compatibility Expansion Manager version: All.
Palm OS version: 4.0 or later.

See “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion* for ways to confirm the presence of the Expansion Manager on the handheld.

See Also [ExpCardInfo\(\)](#), [ExpCardPresent\(\)](#)

ExpSlotMediaType Function

Purpose	Obtains the media type identifier for the specified slot.
Declared In	<code>ExpansionMgr.h</code>
Prototype	<code>SInt32 ExpSlotMediaType (UInt16 slotRefNum, UINT32 *pui32SlotMediaType)</code>
Parameters	<p>→ <i>slotRefNum</i> The slot reference number (passed back by ExpSlotEnumerate()) for which to determine the media type.</p> <p>← <i>pui32SlotMediaType</i> A pointer to a <code>UINT32</code> that identifies the media type of the specified slot. The section “Media Type Constants” on page 552 lists the possible values.</p>
Returns	<p>If successful, returns <code>SYNCERR_NONE</code>.</p> <p>If unsuccessful, returns one of the following error codes:</p> <p><code>expErrCardNotPresent</code> <code>expErrSlotDeallocated</code> <code>expErrUnsupportedOperation</code></p> <p>For descriptions of all error codes, see “Expansion Manager API Error Codes” on page 560.</p>
Compatibility	<p>Expansion Manager version: All. Palm OS version: 4.0 or later.</p> <p>See “Checking for Expansion Cards” on page 59 in the <i>C/C++ Sync Suite Companion</i> for ways to confirm the presence of the Expansion Manager on the handheld.</p>
See Also	ExpCardInfo() , ExpSlotEnumerate()

Expansion Manager API Error Codes

[Table 9.1](#) lists the values of error codes that Expansion Manager API functions can return. The description of each function states which errors each function can return.

All of the named error codes below are defined as preprocessor constants, which are declared in the VFSErr.h header file.

Table 9.1 Expansion Manager API error codes

Value	Code	Description
0x00002901L	expErrUnsupportedOperation	The operation is unsupported or undefined.
0x00002902L	expErrNotEnoughPower	Insufficient battery power on the handheld to perform the operation.
0x00002903L	expErrCardNotPresent	No card is present in the given slot.
0x00002904L	expErrInvalidSlotRefNumber	The slot reference number is not valid.
0x00002905L	expErrSlotDeallocated	The slot reference number is within the valid range, but the Expansion Manager has unloaded the slot driver on the handheld.
0x00002906L	expErrCardNoSectorReadWrite	The card does not support the slot driver block read/write API.
0x0000290AL	expErrNotOpen	The file system library on the handheld necessary for this call has not been installed or has not been opened.
0x0000290DL	expErrEnumerationEmpty	No volumes are present to enumerate or none remain to enumerate.

Virtual File System Manager API

The Virtual File System (VFS) Manager is a layer of software that allows conduits to access all installed file systems on handheld expansion cards. It provides a unified API to conduit 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 Secure Digital (SD), MultiMediaCard (MMC), CompactFlash, Sony’s Memory Stick, and others.

This chapter provides reference material for the VFS Manager API as follows:

VFS Manager Structures and Types	562
VFS Manager Constants	569
VFS Manager Functions	581
VFS Manager Error Codes	645

The VFS Manager functions are available in `VFSAPI.dll` and declared in `VFSMgr.h`. (VFS Manager error codes are declared in `VFSErr.h`.) For more information on the VFS Manager, see [Chapter 6, “Using Expansion Technology,”](#) on page 49 in the *C/C++ Sync Suite Companion*.

NOTE: The VFS Manager is an optional system extension on handhelds. Therefore you should check for the presence of the VFS Manager on the handheld before call any VFS Manager API functions. See [“Verifying Handheld Compatibility”](#) on page 60 in the *C/C++ Sync Suite Companion*.

VFS Manager Structures and Types

This section describes the following data structures and data types that you use with the VFS Manager API.

FileInfoType	Receives information passed back by VFSDirEntryEnumerate() about a file or directory.
FileOrigin	Defines the new position from which to read or write with VFSFileSeek() .
FileRef	Defines references to files and directories.
VFSAnyMountParamType	Defines a base structure for volume mount parameters for different file systems. For slot-based file systems, use the VFSSlotMountParamType structure.
VFSSlotMountParamType	Defines parameters for a card mounted in a physical slot.
VolumeInfoType	Receives information that is passed back by VFSVolumeInfo() and used throughout the VFS Manager functions.

FileInfoType Struct

Purpose Receives information passed back by [VFSDirEntryEnumerate\(\)](#) about a file or directory.

Declared In VFSMgr.h

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](#)” on page 572 for the bits that make up this field.

nameP
Pointer to the buffer that receives the full name of the file or directory. Allocate a sufficiently large buffer and specify its size in nameBufLen.

nameBufLen
Size of the nameP buffer, in bytes.

Compatibility
VFS Manager version: All.
Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also [VFSDirEntryEnumerate\(\)](#)

FileOrigin Typedef

Purpose	Defines the new position from which to read or write with VFSFileSeek() .
Declared In	VFSMgr.h
Prototype	<code>typedef UInt16 FileOrigin</code>
Comments	See “ Seek Origins ” on page 576 for descriptions of the supported file seek origins.
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	VFSFileSeek()

FileRef Typedef

Purpose	Defines references to files and directories.
Declared In	VFSMgr.h
Prototype	<code>typedef UInt32 FileRef</code>
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).

VFSAnyMountParamType Struct

Purpose Defines a base structure for volume mount parameters for different file systems. For slot-based file systems, use the [VFSSlotMountParamType](#) structure.

Declared In VFSMgr.h

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 call [VFSVolumeEnumerate\(\)](#) to successfully enumerate volumes.

reserved
Reserved for future use.

mountClass
Defines the type of mount to use with the specified volume. See “[Volume Mount Classes](#)” on page 579 for a list of mount types.

Compatibility VFS Manager version: All.
Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also [VFSVolumeEnumerate\(\)](#), [VFSSlotMountParamType](#)

Virtual File System Manager API

VFSSlotMountParamType

VFSSlotMountParamType Struct

Purpose Defines parameters for a card mounted in a physical slot.

Declared In VFSMgr.h

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. This is passed back in the [VolumeInfoType](#) structure in a call to [VFSVolumeInfo\(\)](#). Set vfsMountParam->mountClass to VFSMountClass_SlotDriver to mount a physical slot.

slotLibRefNum

Reference number for the slot driver library allocated to the given slot number. If this value is not available, set this field to vfsInvalidSlotLibRefNum.

slotRefNum

The slot reference number obtained by the Expansion Manager's [ExpSlotEnumerate\(\)](#) function.

Comments The VFSSlotMountParamType structure is used when you are mounting a column on a card located in a physical slot. Conduits rely on Palm OS to mount columns, so they use this structure only as a parameter of [VFSVolumeFormat\(\)](#), which can mount a column after a conduit formats it. The vfsMountParam->mountClass field must be set to VFSMountClass_SlotDriver.

Compatibility

VFS Manager version: All.

Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see "[Checking for Expansion Cards](#)" on page 59 in the *C/C++ Sync Suite Companion*).

See Also

[VFSVolumeFormat\(\)](#)

VolumeInfoType Struct

Purpose Receives information that is passed back by [VFSVolumeInfo\(\)](#) and used throughout the VFS Manager functions.

Declared In VFSMgr.h

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](#)” on page 578 for the bits that make up this field.

fsType

File system type for this volume. See “[Defined File Systems](#)” on page 571 for a list of the supported file systems.

fsCreator

Creator code of this volume’s file system driver. This information is used with [VFSCustomControl\(\)](#).

mountClass

Mount class of the driver that mounted this volume. The supported mount classes are listed under “[Volume Mount Classes](#)” on page 579.

slotLibRefNum

Reference to the slot driver library with which the volume is mounted. This field is valid only when the mountClass is `vfsMountClass_SlotDriver`.

slotRefNum

Expansion Manager slot reference number where the card containing the volume is loaded. This field is valid only when the mountClass is `vfsMountClass_SlotDriver`.

mediaType

Type of card media. See “[Media Type Constants](#)” on page 552 for the list of values.

Virtual File System Manager API

VolumeInfoType

`reserved`

Reserved for future use.

Compatibility

VFS Manager version: All.

Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also

[VFSVolumeInfo\(\)](#)

VFS Manager Constants

This section describes the following groups of preprocessor constants that you use with the VFS Manager API.

Date Types	Define the types of dates a conduit can specify with the VFSFileGetDate() and VFSFileSetDate() functions.
Defined File Systems	Define the file systems currently defined by the VFS Manager. These values are used with VFSVolumeInfo() in the VolumeInfoType.fsType parameter.
File and Directory Attributes	Indicate the attributes of a file or directory.
Invalid Reference Numbers	Define placeholders that a caller should use when the it does not have a valid file, slot library, or volume reference number to use.
Miscellaneous Constants	Define miscellaneous constants for the VFS Manager API.
Open Modes	Define the modes in which a file or directory is opened.
Seek Origins	Define file positions to which an offset is added (or subtracted, if the offset is negative) to get a seek position within a file when using VFSFileSeek() .
Versions of the VFS Manager API	Define the major and minor version numbers of the VFS Manager API returned by VFSGetAPIVersion() .
Volume Attributes	Indicate the attributes of a volume.
Volume Mount Classes	Define how a given volume is mounted.

Date Types

Purpose	Define the types of dates a conduit can specify with the VFSFileGetDate() and VFSFileSetDate() functions.
Declared In	VFSMgr.h
Constants	<pre>#define vfsFileDateAccessed 3 Date the file was last accessed. #define vfsFileDateCreated 1 File creation date. #define vfsFileDateModified 2 Date the file was last modified.</pre>
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	VFSFileGetDate() , VFSFileSetDate()

Defined File Systems

Purpose	Define the file systems currently defined by the VFS Manager. These values are used with VFSVolumeInfo() in the VolumeInfoType.fsType parameter.
Declared In	VFSMgr.h
Constants	<pre>#define fsFilesystemType_AFS 'afsu' Unix Andrew file system. #define fsFilesystemType_EXT2 'ext2' Linux file system. #define fsFilesystemType_FAT 'fats' FAT12 and FAT16, which handles only 8.3 filenames. #define fsFilesystemType_FFS 'ffsb' Unix Berkeley block based file system. #define fsFilesystemType_HFS 'hfss' Macintosh standard hierarchical file system. #define fsFilesystemType_HFSPlus 'hfse' Macintosh extended hierarchical file system. #define fsFilesystemType_HPFS 'hpfs' OS/2 High Performance file system #define fsFilesystemType_MFS 'mfso' Macintosh original file system. #define fsFilesystemType_NFS 'nfsu' Unix Networked file system. #define fsFilesystemType_Novell 'novl' Novell file system. #define fsFilesystemType_NTFS 'ntfs' Windows NT file system. #define fsFilesystemType_VFAT 'vfat' FAT12 and FAT16, extended to handle long filenames.</pre>
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see " Checking for Expansion Cards " on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	VFSVolumeInfo()

Virtual File System Manager API

File and Directory Attributes

File and Directory Attributes

Purpose	Indicate the attributes of a file or directory.
Declared In	VFSMgr.h
Constants	<pre>#define vfsFileAttrArchive (0x00000020UL) Archived file or directory. #define vfsFileAttrDirectory (0x00000010UL) A directory, not a file. #define vfsFileAttrHidden (0x00000002UL) Hidden file or directory. #define vfsFileAttrLink (0x00000040UL) Link to another file or directory. #define vfsFileAttrReadOnly (0x00000001UL) Read-only file or directory. #define vfsFileAttrSystem (0x00000004UL) System file or directory. #define vfsFileAttrVolumeLabel (0x00000008UL) Volume label.</pre>
Comments	Use these attributes 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.
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	VFSFileGetAttributes() , VFSFileSetAttributes()

Invalid Reference Numbers

Purpose	Define placeholders that a caller should use when it does not have a valid file, slot library, or volume reference number to use.
Declared In	VFSMgr.h
Constants	<pre>#define vfsInvalidFileRef 0 The file reference number is invalid. #define vfsInvalidSlotLibRefNum (-1) The slot library reference number is not available. See the description of the VFSSlotMountParamType structure. #define vfsInvalidVolRef 0 The volume has not been formatted. See the description of VFSVolumeFormat().</pre>
Comments	These placeholder values are guaranteed not to represent a valid reference number. Use them like you would use NULL for a file pointer.
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	VFSSlotMountParamType , VFSVolumeFormat()

Miscellaneous Constants

Purpose	Define miscellaneous constants for the VFS Manager API.
Declared In	VFSMgr.h
Constants	<pre>#define sysFileCVFSMgr 'vfsm' Defines the creator ID for the VFS Manager in the device's ROM. You can use this with SyncReadFeature() to get the version number of the VFS Manager. However, the preferred way to get the version number is to call VFSSupport(). #define vfsFtrIDVersion 0 Feature number used to obtain the version of the VFS Manager in the device's ROM. Use this number in conjunction with a creator ID of sysFileCVFSMgr.</pre>

Open Modes

Purpose	Define the modes in which a file or directory is opened.
Declared In	VFSMgr.h
Constants	<pre>#define vfsModeCreate (0x0008UL) Create the file if it doesn't already exist. #define vfsModeExclusive (0x0001UL) Open and lock the file or directory. This mode excludes anyone else from using the file or directory until it is closed. #define vfsModeRead (0x0002UL) Open for read access. #define vfsModeReadWrite (vfsModeWrite vfsModeRead) Open for read/write access. #define vfsModeTruncate (0x0010UL) Truncate the file to zero bytes after opening, removing all existing data. #define vfsModeWrite (0x0004UL vfsModeExclusive) Open for exclusive write access. This mode excludes anyone else from using the file or directory until it is closed.</pre>
Comments	Specify one of these values in the <i>openMode</i> parameter of VFSFileOpen() .
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	VFSFileOpen()

Seek Origins

Purpose Define file positions to which an offset is added (or subtracted, if the offset is negative) to get a seek position within a file when using [VFSFileSeek\(\)](#).

Declared In VFSMgr.h

Constants

```
#define fsOriginBeginning 0
    From the beginning (first data byte of file).
#define fsOriginCurrent 1
    From the current position.
#define fsOriginEnd 2
    From the end of the file (one position beyond the last data
    byte). Only negative offsets are legal from this origin.
```

Compatibility VFS Manager version: All.
Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also [VFSFileSeek\(\)](#)

Versions of the VFS Manager API

Purpose Define the major and minor version numbers of the VFS Manager API returned by [VFSGetAPIVersion\(\)](#).

Declared In VFSMgr.h

Constants

```
#define VFSAPI_VER_MAJOR_1 1  
      Indicates the major version number is 1.  
  
#define VFSAPI_VER_MINOR_0 0  
      Indicates the minor version number is 0.  
  
#define VFSAPI_VER_MINOR_1 1  
      Indicates the minor version number is 1.
```

Volume Attributes

Purpose	Indicate the attributes of a volume.
Declared In	VFSMgr.h
Constants	<pre>#define vfsVolumeAttrHidden (0x00000002UL) The volume should not be visible to the user. For more information, see "Hidden Volumes" on page 68 in the C/C++ Sync Suite Companion.</pre> <pre>#define vfsVolumeAttrReadOnly (0x00000002UL) The volume is read only.</pre> <pre>#define vfsVolumeAttrSlotBased (0x00000001UL) The volume is associated with a slot driver as opposed to the Palm OS® Emulator.</pre>
Comments	Use these attributes individually or in combination when interpreting the attributes field of a VolumeInfoType structure passed back by VFSVolumeInfo() .
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see " Checking for Expansion Cards " on page 59 in the C/C++ Sync Suite Companion).
See Also	VFSVolumeInfo() , VolumeInfoType

Volume Mount Classes

Purpose	Define how a given volume is mounted.
Declared In	VFSMgr.h
Constants	#define vfsMountClass_Simulator sysFileTSimulator Mount the volume through the 68K Palm Simulator. This is used for testing. #define vfsMountClass_SlotDriver sysFileTSlotDriver Mount the volume with a slot driver shared library. #define sysFileTSimulator '\?\?\?\?' File type for 68K Palm Simulator files (app.tres, sys.tres). vfsMountClass_Simulator is defined as this value. #define sysFileTSlotDriver 'libs' File type for slot driver libraries. vfsMountClass_SlotDriver is defined as this value.
Comments	The mountClass fields in the VFSAnyMountParamType and VolumeInfoType structures take one of the vfsMount... values, two of which are defined as the sysFileT... values also listed above.
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	VFSAnyMountParamType , VolumeInfoType

Volume Format/Mount Flags

Purpose	Flags that control how a volume is mounted after formatting.
Declared In	VFSMgr.h
Constants	<pre>#define vfsMountFlagsReserved1 0x08 Reserved for future use. #define vfsMountFlagsReserved2 0x10 Reserved for future use. #define vfsMountFlagsReserved3 0x20 Reserved for future use. #define vfsMountFlagsReserved4 0x40 Reserved for future use. #define vfsMountFlagsReserved5 0x80 Reserved for future use. #define vfsMountFlagsUseThisFileSystem 0x01 Pass this flag to cause the volume to be mounted or formatted using the file system specified by the specified file system.</pre>
Comments	<p>Volumes are mounted as part of the volume format process using VFSVolumeFormat().</p> <p>Pass no flags (0) to have the VFS Manager attempt to mount or format the volume using a file system appropriate to the slot.</p>

VFS Manager Functions

This section describes the following functions, which enable conduits to access file systems on expansion cards.

<u>VFSCustomControl()</u>	Makes a custom API call to a particular file system driver, given the driver's creator ID.
<u>VFSDirCreate()</u>	Creates a new directory.
<u>VFSDirEntryEnumerate()</u>	Enumerates the entries in a given directory. Entries can include files, links, and other directories.
<u>VFSExportDatabaseToFile()</u>	Flattens and exports the specified database on the handheld to the specified PDB or PRC file on an expansion card. Works only with classic databases.
<u>VFSExportDatabaseToFileEx()</u>	Flattens and exports the specified database on the handheld to the specified image file on an expansion card. Works with schema, extended, and classic databases only on handhelds running Palm OS Cobalt, version 6.0.1 or later.
<u>VFSfileClose()</u>	Closes an opened file or directory.
<u>VFSfileCreate()</u>	Creates a file given a volume reference number and a path.
<u>VFSfileDelete()</u>	Deletes a closed file or directory.
<u>VFSfileEOF()</u>	Gets end-of-file status for an open file.
<u>VFSfileGet()</u>	Reads the file from the expansion card on the handheld and copies it to the desktop.
<u>VFSfileGetAttributes()</u>	Gets the attributes of an open file or directory.
<u>VFSfileGetDate()</u>	Gets the dates of an open file or directory.
<u>VFSfileOpen()</u>	Opens a file or directory and returns a reference pointer to it.

Virtual File System Manager API

VFS Manager Functions

<u>VFSFilePut()</u>	Reads a file from the desktop and copies it to an expansion card on the handheld.
<u>VFSFileRead()</u>	Reads data from a file into the specified buffer.
<u>VFSFileRename()</u>	Renames a closed file or directory.
<u>VFSFileResize()</u>	Changes the size of an open file.
<u>VFSFileSetAttributes()</u>	Sets the position from which to read or write within an open file.
<u>VFSFileSetDate()</u>	Sets the attributes of an open file or directory.
<u>VFSFileSeek()</u>	Changes the dates of an open file or directory.
<u>VFSFileSize()</u>	Gets the size of an open file.
<u>VFSFileTell()</u>	Gets the current position of the file pointer within an open file.
<u>VFSFileWrite()</u>	Writes data to an open file.
<u>VFSGetAPIVersion()</u>	Retrieves the version of the Expansion Manager and VFS Manager APIs.
<u>VFSGetDefaultDirectory()</u>	Retrieves the default directory on the given volume for files of a particular type.
<u>VFSImportDatabaseFromFile()</u>	Creates a database from the specified PDB or PRC file on an expansion card. Works only with classic databases.
<u>VFSImportDatabaseFromFileEx()</u>	Creates a database from the specified image file on an expansion card. Works with schema, extended, and classic databases only on handhelds running Palm OS Cobalt, version 6.0.1 or later.
<u>VFSSupport()</u>	Determines whether the Expansion Manager and VFS Manager are present on the handheld, their version if present, and gets expansion slot and volume information.
<u>VFSVolumeEnumerate()</u>	Enumerates the mounted volumes and retrieves a list of volume reference numbers.

<u>VFSVolumeFormat()</u>	Formats and mounts the first volume installed in a given slot.
<u>VFSVolumeGetLabel()</u>	Gets the volume label for a particular volume.
<u>VFSVolumeInfo()</u>	Gets information about the specified volume.
<u>VFSVolumeSetLabel()</u>	Changes the volume label for a mounted volume.
<u>VFSvolumeSize()</u>	Determines the total amount of space on a volume, as well as the amount that is currently being used.

VFSCustomControl Function

Purpose	Makes a custom API call to a particular file system driver, given the driver's creator ID.
Declared In	VFSMgr.h
Prototype	SInt32 VFSCustomControl (UInt32 <i>fsCreator</i> , UInt32 <i>apiCreator</i> , UInt16 <i>apiSelector</i> , void * <i>pDataBuf</i> , UInt16 * <i>pwDataBufLen</i>)
Parameters	<p>→ <i>fsCreator</i> Creator of the file system on the handheld to call. A value of zero tells the VFS Manager to check each registered file system, looking for one that supports the call.</p> <p>→ <i>apiCreator</i> Registered creator ID of the file system driver on the handheld.</p> <p>→ <i>apiSelector</i> Code for the custom operation that you want the file system driver to perform. See the file system driver manufacturer for details.</p> <p>↔ <i>pDataBuf</i> 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 <i>pwDataBufLen</i>, the contents of this buffer may have been updated.</p> <p>↔ <i>pwDataBufLen</i> On entry, points to the size of the <i>pDataBuf</i> buffer. On exit, this value reflects the size of the data written to the <i>pDataBuf</i> buffer. If <i>pwDataBufLen</i> is NULL, <i>pDataBuf</i> is passed to the file system but is not updated on exit.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: expErrNotOpen expErrUnsupportedOperation SYNCERR_COMM_NOT_INIT SYNCERR_REMOTE_BAD_ARG vfsErrInvalidOperation

`vfsErrNoFileSystem`

For descriptions of all VFS Manager error codes, see “[VFS Manager Error Codes](#)” on page 645.

Comments

The driver identifies the call and its API by a registered creator ID and a selector. (You can use [VFSVolumeInfo\(\)](#) to determine the creator ID of the file system for a given volume.) 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

VFS Manager version: All.

Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also

[VFSVolumeInfo\(\)](#)

Virtual File System Manager API

VFSDirCreate

VFSDirCreate Function

Purpose	Creates a new directory.
Declared In	<code>VFSMgr.h</code>
Prototype	<code>SInt32 VFSDirCreate (UInt16 volRefNum, const char *pszDirName)</code>
Parameters	<p>→ <i>volRefNum</i> Volume reference number passed back by VFSVolumeEnumerate().</p> <p>→ <i>pszDirName</i> A pointer to the full path of the directory to be created.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error codes:</p> <p><code>expErrNotOpen</code> <code>SYNCERR_COMM_NOT_INIT</code> <code>SYNCERR_REMOTE_BAD_ARG</code> <code>vfsErrBadName</code> <code>vfsErrFileAlreadyExists</code> <code>vfsErrInvalidOperation</code> <code>vfsErrNoFileSystem</code> <code>vfsErrVolumeBadRef</code> <code>vfsErrVolumeFull</code></p> <p>For descriptions of all VFS Manager error codes, see “VFS Manager Error Codes” on page 645.</p>
Comments	<p>All parts of the path except the last component must already exist. The <code>vfsFileAttrDirectory</code> attribute is set with this function.</p> <p>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().</p>
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager

is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also [VFSFileOpen\(\)](#), [VFSFileDelete\(\)](#),
[VFSDirEntryEnumerate\(\)](#)

Virtual File System Manager API

VFSDirEntryEnumerate

VFSDirEntryEnumerate Function

Purpose	Enumerates the entries in a given directory. Entries can include files, links, and other directories.
Declared In	VFSMgr.h
Prototype	SInt32 VFSDirEntryEnumerate (FileRef <i>dirRef</i> , UIInt32 * <i>pui32DirEntryIterator</i> , FileInfoType * <i>pFileInfo</i>)
Parameters	<p>→ <i>dirRef</i> Directory reference passed back by VFSFileOpen().</p> <p>↔ <i>pui32DirEntryIterator</i> Pointer to the index of the last entry enumerated. For the first iteration, initialize this parameter to the constant <code>expIteratorStart</code>. Upon return, this references the next entry in the directory. If <i>pFileInfo</i> is the last entry, this parameter is set to <code>expIteratorStop</code>.</p> <p>← <i>pFileInfo</i> Pointer to the FileInfoType data structure that contains information about the given directory entry.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: <code>expErrEnumerationEmpty</code> <code>expErrNotOpen</code> <code>SYNCERR_COMM_NOT_INIT</code> <code>SYNCERR_REMOTE_BAD_ARG</code> <code>vfsErrFileBadRef</code> <code>vfsErrInvalidOperation</code> <code>vfsErrNotADirectory</code> <code>vfsErrNoFileSystem</code> For descriptions of all VFS Manager error codes, see “ VFS Manager Error Codes ” on page 645.
Comments	The file system function returns information on the entry referenced by <i>pui32DirEntryIterator</i> . The directory to be enumerated must first be opened with VFSFileOpen() to obtain a file reference

number. To get 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 `expIteratorStart` and `expIteratorStop`. Before the first call to `VFSDirEntryEnumerate()`, initialize `pui32DirEntryIterator` to the constant value `expIteratorStart`. Each iteration then increments the value pointed to by `pui32DirEntryIterator` to the next entry. When this function returns information on the last entry in the directory, `pui32DirEntryIterator` is set to `expIteratorStop`.

IMPORTANT: Creating, renaming, or deleting any file or directory invalidates the enumeration. After any such operation, the enumeration will need to be restarted.

Virtual File System Manager API

VFSDirEntryEnumerate

Example The following illustrates how to use `VFSDirEntryEnumerate()`.

```
// Open the directory and iterate through the files in it.  
// volRefNum must have already been defined.  
FileRef dirRef;  
  
err = VFSFileOpen (volRefNum, "/", vfsModeRead, &dirRef);  
if(err == errNone) {  
    // Iterate through all the files in the open directory  
    UInt32 fileIterator;  
    FileInfoType fileInfo;  
    char *fileName = new char[256]; // Should check for err.  
  
    fileInfo.nameP = fileName; // Point to local buffer.  
    fileInfo.nameBufLen = sizeof(fileName);  
    fileIterator = expIteratorStart;  
    while (fileIterator != expIteratorStop) {  
        // Get the next file  
        err = VFSDirEntryEnumerate (dirRef, &fileIterator,  
            &fileInfo);  
        if(err == errNone) {  
            // Process the file here.  
        }  
        else {  
            // Handle directory open error here.  
        }  
        delete [] fileName;  
    }  
}
```

Compatibility

VFS Manager version: All.

Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also

[VFSFileOpen\(\)](#), “[Directory Enumeration Constants](#)” on page 550

VFSExportDatabaseToFile Function

Purpose	Flattens and exports the specified database on the handheld to the specified PDB or PRC file on an expansion card. Works only with classic databases.
Declared In	<code>VFSMgr.h</code>
Prototype	<pre>SInt32 VFSExportDatabaseToFile (UInt16 volRefNum, const char *pszPathName, UInt16 wCardNumber, LocalID dbID)</pre>
Parameters	<p>→ <i>volRefNum</i> Volume reference number (passed back by VFSVolumeEnumerate()) of the volume on which to create the destination file.</p> <p>→ <i>pszPathName</i> Pointer to the full path and filename of the destination file to create. All parts of the path, excluding the filename, must already exist.</p> <p>→ <i>wCardNumber</i> RAM card number in the handheld on which the database exists. Note that this does not refer to the expansion card and is therefore not related to the slot reference number. The card number for the first RAM memory card on the handheld is 0, which is the only one that most handhelds have.</p> <p>→ <i>dbID</i> The local ID of the database on the handheld. This is the dwLocalID field of the SyncDatabaseInfoType structure, which is passed back by the following functions: SyncFindDbByName(), SyncFindDbByTypeCreator(), and SyncOpenDB().</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: SYNCERR_COMM_NOT_INIT SYNCERR_REMOTE_BAD_ARG vfsErrBadName vfsErrInvalidOperation

Virtual File System Manager API

VFSExportDatabaseToFile

For descriptions of all VFS Manager error codes, see “[VFS Manager Error Codes](#)” on page 645.

Comments This utility function flattens and exports a database from primary storage memory on a handheld to a PDB or PRC file on an expansion card. This function is the opposite of [VFSSImportDatabaseFromFile\(\)](#). Use this function, for example, to copy applications from primary storage to an expansion card.

IMPORTANT: With VFS Manager API version 1.1, this function is deprecated for use with handhelds running Palm OS Cobalt; call [VFSExportDatabaseToFileEx\(\)](#) instead. The reason for replacing this function is that memory card numbers are not supported by Palm OS Cobalt and local IDs are insufficient to identify schema databases and are cumbersome to identify extended and classic databases.

However, this function is still supported for use with Palm OS 4.x and Palm OS Garnet.

Example The following example illustrates how to use `VFSExportDatabaseToFile()` to export the MemoPad database.

```
SyncFindDbByTypeCreatorParams rParam;
SyncDatabaseInfoType rInfo;

memset (&rParam, 0, sizeof (rParam));
rParam.bSrchFlags = SYNC_DB_SRCH_OPT_NEW_SEARCH;
rParam.dwCreator = 'memo';

SyncFindDbByTypeCreator (rParam, rInfo);
SInt32 retval = VFSExportDatabaseToFile (volRefNum,
                                         "/Palm/Launcher/Memopad.pdb", 0, rInfo.dwLocalId);
```

Compatibility VFS Manager version: All.
Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also [VFSFileWrite\(\)](#), [VFSSImportDatabaseFromFile\(\)](#)

VFSExportDatabaseToFileEx Function

Flattens and exports the specified database on the handheld to the specified image file on an expansion card. Works with schema, extended, and classic databases only on handhelds running Palm OS Cobalt, version 6.0.1 or later.

Declared In

VFSMgr.h

Prototype

```
SInt32 VFSExportDatabaseToFileEx  
    (UInt16 volRefNum, char *pszPathName,  
     DBDatabaseInfo *pDbInfo)
```

Parameters

→ *volRefNum*

Volume reference number (passed back by [VFSVolumeEnumerate\(\)](#)) of the volume on which to create the destination file.

↔ *pszPathName*

Pointer to the full path and filename or only to the directory of the destination file to create. Do not specify NULL. All parts of the path, excluding the filename, must already exist. If you specify a directory, this function automatically generates the filename and appends it to the directory name. Upon return, receives the full path and filename to which the database is written.

→ *pDbInfo*

A pointer to a [DBDatabaseInfo](#) structure whose name, creator, type and attributes fields specify the database to export. The type is only used as a cross-check and may be set to zero if you don't care what its value is. The attributes must specify whether the database is a classic, extended, or schema database. All other attributes bits are ignored.

Returns

If successful, returns 0.

If unsuccessful, returns one of the following error codes:

SYNCERR_COMM_NOT_INIT

SYNCERR_REMOTE_BAD_ARG

SYNCERR_UNKNOWN_REQUEST

The handheld is running a version of Palm OS that this function does not support. See "Compatibility" below.

vfsErrBadName

Virtual File System Manager API

VFSExportDatabaseToFileEx

`vfsErrInvalidOperation`

For descriptions of all VFS Manager error codes, see “[VFS Manager Error Codes](#)” on page 645.

Comments

This utility function flattens and exports a schema, extended, or classic database from primary storage memory on a handheld to a to an image file on an expansion card. This function does the opposite of [VFSImportDatabaseFromFileEx\(\)](#). Use this function, for example, to copy applications or databases from primary storage to an expansion card.

With this function, you identify a database to export by the creator, name, attributes, type fields of the input DBDatabaseInfo structure. Then you specify the location to which to export the file: the volume and a path and filename or only a directory path. If you specify only a directory, then the Sync Manager generates a unique filename in the same way it does for [SyncGenerateBackupFileName\(\)](#) and appends it to the directory path and passes back the full path and filename when this function returns.

This function is a replacement for [VFSExportDatabaseToFile\(\)](#), which identifies databases only by memory card number and local ID. The reason for replacing this function is that memory card numbers are not supported by Palm OS Cobalt and local IDs are insufficient to identify schema databases and somewhat cumbersome to identify extended and classic databases.

Example

The following example illustrates how to use VFSExportDatabaseToFileEx() to export a given schema database.

```
UInt16 volRefNum; // Get this via VFSVolumeEnumerate().
char pathName[128] = "/Palm/Launcher/MySchemaDatabase.sdb";
DBDatabaseInfo dbInfo;
dbInfo.name = "MySchemaDatabase";
dbInfo.creator = 0x0017;
dbInfo.type = 0;
dbInfo.attributes = dmHdrAttrSchema;

SInt32 retval = VFSExportDatabaseToFileEX (volRefNum,
    pathName, &dbInfo);
```

Compatibility	VFS Manager version: 1.1 or later. Palm OS version: 6.0.1 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	DBDatabaseInfo , VFSImportDatabaseFromFileEx()

Virtual File System Manager API

VFSFileClose

VFSFileClose Function

Purpose	Closes an opened file or directory.
Declared In	VFSMgr.h
Prototype	SInt32 VFSFileClose (FileRef <i>fileRef</i>)
Parameters	→ <i>fileRef</i> File reference number passed back from VFSFileOpen() .
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: expErrNotOpen SYNCERR_COMM_NOT_INIT vfsErrFileBadRef vfsErrInvalidOperation For descriptions of all VFS Manager error codes, see “ VFS Manager Error Codes ” on page 645.
Comments	VFSFileClose() closes a file or directory that has been opened with VFSFileOpen() .
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	VFSFileOpen()

VFSFileCreate Function

Purpose	Creates a file given a volume reference number and a path.
Declared In	VFSMgr.h
Prototype	SInt32 VFSFileCreate (UInt16 <i>volRefNum</i> , const char * <i>pszPathName</i>)
Parameters	<p>→ <i>volRefNum</i> Volume reference number (passed back by VFSVolumeEnumerate() of the volume on which to create the file.</p> <p>→ <i>pszPathName</i> Pointer to the full path of the file to be created. All parts of the path, excluding the filename, must already exist.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error codes:</p> <p>expErrNotOpen SYNCERR_COMM_NOT_INIT SYNCERR_REMOTE_BAD_ARG vfsErrBadName vfsErrFileAlreadyExists vfsErrInvalidOperation vfsErrNoFileSystem vfsErrVolumeBadRef vfsErrVolumeFull</p> <p>For descriptions of all VFS Manager error codes, see “VFS Manager Error Codes” on page 645.</p>
Comments	<p>All parts of the path except the last component must already exist. VFSFileCreate() does not open the file. Any operations you want to perform on this directory require a reference, which is obtained through a call to VFSFileOpen().</p> <p>It is the responsibility of the file system library on the handheld 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</p>

Virtual File System Manager API

VFSFileCreate

convention for a FAT file system without long filename support. See “[Directory Paths](#)” on page 74 in the *C/C++ Sync Suite Companion* for a description of how to construct a valid path.

This function does not open the file. [VFSFileOpen\(\)](#) must be used to open the file. Neither does it create a directory. To create a directory use [VFSDirCreate\(\)](#).

Compatibility

VFS Manager version: All.

Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also

[VFSFileDelete\(\)](#), [VFSFileOpen\(\)](#), [VFSDirCreate\(\)](#)

VFSFileDelete Function

Purpose	Deletes a closed file or directory.
Declared In	VFSMgr.h
Prototype	SInt32 VFSFileDelete (UInt16 volRefNum, const char *pszPathName)
Parameters	<p>→ <i>volRefNum</i> Volume reference number (passed back by VFSVolumeEnumerate()) of the volume on which to delete the file.</p> <p>→ <i>pszPathName</i> Pointer to the full path of the file or directory to delete.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: expErrNotOpen SYNCERR_COMM_NOT_INIT SYNCERR_REMOTE_BAD_ARG vfsErrBadName vfsErrDirNotEmpty vfsErrFileNotFoundException vfsErrFilePermissionDenied vfsErrFileStillOpen vfsErrInvalidOperation vfsErrNoFileSystem vfsErrVolumeBadRef For descriptions of all VFS Manager error codes, see “ VFS Manager Error Codes ” on page 645.
Comments	A directory must be empty before VFSFileDelete() can delete it.
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager

Virtual File System Manager API

VFSFileDelete

is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also [`VFSFileCreate\(\)`](#), [`VFSDirCreate\(\)`](#), [`VFSFileClose\(\)`](#)

VFSFileEOF Function

Purpose	Gets end-of-file status for an open file.
Declared In	VFSMgr.h
Prototype	SInt32 VFSFileEOF (FileRef <i>fileRef</i>)
Parameters	→ <i>fileRef</i> File reference number passed back by VFSFileOpen() .
Returns	If successful, returns 0 (the file pointer was not already at the end of the file). If unsuccessful, returns one of the following error codes: expErrNotOpen SYNCERR_COMM_NOT_INIT vfsErrFileBadRef vfsErrFileEOF vfsErrInvalidOperation vfsErrIsADirectory vfsErrNoFileSystem For descriptions of all VFS Manager error codes, see “ VFS Manager Error Codes ” on page 645.
Comments	This function operates only on files and cannot be used with directories.
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	VFSFileOpen()

Virtual File System Manager API

VFSFileGet

VFSFileGet Function

Purpose	Reads the file from the expansion card on the handheld and copies it to the desktop.
Declared In	VFSMgr.h
Prototype	SInt32 VFSFileGet (UInt16 <i>volRefNum</i> , const char * <i>pszDevicePathName</i> , const char * <i>pszDiskPathName</i>)
Parameters	<p>→ <i>volRefNum</i> Volume reference number (passed back by VFSVolumeEnumerate()) of the volume on which the file is present.</p> <p>→ <i>pszDevicePathName</i> Full path and filename of the file to be read from the expansion card on the handheld.</p> <p>→ <i>pszDiskPathName</i> Full path and filename for the file to be created on the desktop. All parts of the path, except the file, must already exist. If the file does not exist, then this function creates it. If the file exists, then it overwrites the file.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: expErrNotOpen SYNCERR_COMM_NOT_INIT SYNCERR_REMOTE_BAD_ARG vfsErrDisk FileAccess vfsErr FileAccess Other vfsErr FileAccess Bad Ref vfsErr FileAccess EOF vfsErr FileAccess Permission Denied vfsErr Invalid Operation vfsErr Is A Directory vfsErr No File System

For descriptions of all VFS Manager error codes, see “[VFS Manager Error Codes](#)” on page 645.

Comments Using `VFSFileGet()` to copy a file to the desktop is easier than opening the file on the expansion card and reading it into a buffer on the desktop.

Compatibility VFS Manager version: All.
Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also [VFSFileOpen\(\)](#), [VFSFilePut\(\)](#)

Virtual File System Manager API

VFSFileGetAttributes

VFSFileGetAttributes Function

Purpose	Gets the attributes of an open file or directory.
Declared In	VFSMgr.h
Prototype	SInt32 VFSFileGetAttributes (FileRef <i>fileRef</i> , UIInt32 * <i>pui32Attributes</i>)
Parameters	<p>→ <i>fileRef</i> File reference number passed back by VFSFileOpen().</p> <p>← <i>pui32Attributes</i> Pointer to the attributes of the file or directory. See “File and Directory Attributes” on page 572 for a list of values that can be passed back through this parameter.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: expErrNotOpen SYNCERR_COMM_NOT_INIT vfsErrFileBadRef vfsErrInvalidOperation vfsErrNoFileSystem For descriptions of all VFS Manager error codes, see “ VFS Manager Error Codes ” on page 645.
Comments	The file or directory must be open before calling this function.
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	VFSFileOpen() , VFSFileGetDate() , VFSFileSetAttributes()

VFSFileGetDate Function

Purpose	Gets the dates of an open file or directory.
Declared In	VFSMgr.h
Prototype	SInt32 VFSFileGetDate (FileRef <i>fileRef</i> , UInt16 <i>whichDate</i> , UInt32 * <i>pui32Date</i>)
Parameters	<p>→ <i>fileRef</i> File reference number passed back by VFSfileOpen().</p> <p>→ <i>whichDate</i> Specifies which date—creation, modification, or last access—you want. Supply one of the values described in “Date Types” on page 570.</p> <p>← <i>pui32Date</i> 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.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: expErrNotOpen expErrUnsupportedOperation SYNCERR_COMM_NOT_INIT SYNCERR_REMOTE_BAD_ARG vfsErrFileBadRef vfsErrInvalidOperation vfsErrNoFileSystem For descriptions of all VFS Manager error codes, see “ VFS Manager Error Codes ” on page 645.
Comments	The file or directory must be open before calling this function. Note that not all file systems are required to support all date types. If the supplied date type is not supported by the file system, VFSFileGetDate returns expErrUnsupportedOperation.
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager

Virtual File System Manager API

VFSFileGetDate

is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also [VFSFileOpen\(\)](#), [VFSFileGetAttributes\(\)](#),
[VFSFileSetDate\(\)](#)

VFSFileOpen Function

Purpose	Opens a file or directory and returns a reference pointer to it.
Declared In	VFSMgr.h
Prototype	SInt32 VFSFileOpen (UInt16 <i>volRefNum</i> , const char * <i>pszPathName</i> , UInt16 <i>openMode</i> , FileRef * <i>pFileRef</i>)
Parameters	<p>→ <i>volRefNum</i> Volume reference number (passed back by VFSVolumeEnumerate()) of the volume on which to open the file.</p> <p>→ <i>pszPathName</i> Pointer to the full path of the file or directory to be opened. This must be a valid path. It cannot be empty and cannot contain null characters. The format of the path should match what the underlying file system supports. See “Directory Paths” on page 74 in the <i>C/C++ Sync Suite Companion</i> for a description of how to construct a valid path.</p> <p>→ <i>openMode</i> Mode to use when opening the file. See “Open Modes” on page 575 for a list of accepted modes.</p> <p>← <i>pFileRef</i> Pointer to the opened file or directory.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: expErrNotOpen SYNCERR_COMM_NOT_INIT SYNCERR_REMOTE_BAD_ARG vfsErrBadName vfsErrFileNotFoundException vfsErrFilePermissionDenied vfsErrInvalidOperation vfsErrVolumeBadRef

Virtual File System Manager API

VFSFileOpen

For descriptions of all VFS Manager error codes, see “[VFS Manager Error Codes](#)” on page 645.

Comments	The file reference number (<i>pFileRef</i>) obtained for a directory cannot be used for all functions. For example, it is not permitted (or logical) to read directly from an opened directory.
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	VFSFileClose() , VFSDirEntryEnumerate()

VFSFilePut Function

Purpose	Reads a file from the desktop and copies it to an expansion card on the handheld.
Declared In	<code>VFSMgr.h</code>
Prototype	<pre>SInt32 VFSFilePut (UInt16 volRefNum, const char *pszDevicePathName, const char *pszDiskPathName)</pre>
Parameters	<p>→ <i>volRefNum</i> Volume reference number (passed back by VFSVolumeEnumerate()) of the volume on which to put the file.</p> <p>→ <i>pszDevicePathName</i> Full path and filename of the destination file on the handheld. All parts of the path, except the file, must exist. Can also be set to NULL (see “Comments” below).</p> <p>→ <i>pszDiskPathName</i> Full path and filename for the file to be read from the desktop.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: <code>expErrNotOpen</code> <code>SYNCERR_COMM_NOT_INIT</code> <code>SYNCERR_REMOTE_BAD_ARG</code> <code>vfsErrBadName</code> <code>vfsErrDirectoryNotFound</code> <code>vfsErrDisk FileAccess</code> <code>vfsErr FileAccess Other</code> <code>vfsErrFileAlreadyExists</code> <code>vfsErrFileNotFoundException</code> <code>vfsErrFilePermissionDenied</code> <code>vfsErrInvalidOperation</code> <code>vfsErrNoFileSystem</code>

Virtual File System Manager API

VFSFilePut

vfsErrVolumeBadRef

vfsErrVolumeFull

For descriptions of all VFS Manager error codes, see “[VFS Manager Error Codes](#)” on page 645.

Comments The behavior of this function depends on whether a destination path is specified:

- If *pszDevicePathName* is specified, all parts of the path, except the filename, must already exist.
 - If the full path exists, this function copies the file to specified location.
 - If the full path does *not* exist, this function fails and returns an error.
- If *pszDevicePathName* is NULL or points to an empty string:
 - If a default directory is registered for this file type, the VFS Manager ensures that the entire path exists—creating the directories leading up to the default directory, if necessary—and puts the file in the default directory.
 - If no default directory is registered for this file type, this function returns *vfsErrDirectoryNotFound*.

If the path exists in either of the above cases, this function copies the file specified by *pszDiskPathName* to the destination on the expansion card. If the file already exists at the destination, this function overwrites it with the one specified by *pszDiskPathName*.

Compatibility VFS Manager version: All.

Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also [VFSGetDefaultDirectory\(\)](#), [VFSFileGet\(\)](#)

VFSFileRead Function

Purpose	Reads data from a file into the specified buffer.
Declared In	VFSMgr.h
Prototype	SInt32 VFSFileRead (FileRef <i>fileRef</i> , UInt32 <i>numBytes</i> , void * <i>pBuffer</i> , UInt32 * <i>pNumBytesRead</i>)
Parameters	<p>→ <i>fileRef</i> File reference number passed back by VFSFileOpen().</p> <p>→ <i>numBytes</i> Number of bytes to read.</p> <p>← <i>pBuffer</i> A pointer to the destination memory chunk on the desktop where the data is stored. The caller must preallocate this buffer to hold at least <i>numBytes</i> characters.</p> <p>← <i>pNumBytesRead</i> A 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.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error codes:</p> <p>expErrNotOpen SYNCERR_COMM_NOT_INIT vfsErrFileBadRef vfsErrFileEOF vfsErrFilePermissionDenied vfsErrInvalidOperation vfsErrIsADirectory vfsErrNoFileSystem</p> <p>For descriptions of all VFS Manager error codes, see “VFS Manager Error Codes” on page 645.</p>

Virtual File System Manager API

VFSFileRead

Comments	This function operates only on files and cannot be used with directories; use VFSDirEntryEnumerate() to explore the contents of a directory.
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	VFSFileWrite() , VFSImportDatabaseFromFile()

VFSFileRename Function

Purpose	Renames a closed file or directory.
Declared In	VFSMgr.h
Prototype	SInt32 VFSFileRename (UInt16 volRefNum, const char *pszPathName, const char *pszNewName)
Parameters	<p>→ <i>volRefNum</i> Volume reference number (passed back by VFSVolumeEnumerate()) of the volume on which to rename the file.</p> <p>→ <i>pszPathName</i> Pointer to the full path of the file or directory to rename.</p> <p>→ <i>pszNewName</i> Pointer to the new filename. Note that this is the name of the file only and does not include the path to the file.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: expErrNotOpen SYNCERR_COMM_NOT_INIT SYNCERR_REMOTE_BAD_ARG vfsErrBadName vfsErrFileAlreadyExists vfsErrFileNotFoundException vfsErrFilePermissionDenied vfsErrFileStillOpen vfsErrInvalidOperation vfsErrNoFileSystem vfsErrVolumeBadRef vfsErrVolumeFull For descriptions of all VFS Manager error codes, see “ VFS Manager Error Codes ” on page 645.

Virtual File System Manager API

VFSFileRename

Comments	This function cannot be used to move a file to another location within the file system. This function returns <code>vfsErrBadName</code> if either <code>pszPathName</code> or <code>pszNewName</code> is invalid, or if the string pointed to by <code>pszNewName</code> is a path rather than a filename.
Example	Below is an example of how to use <code>VFSFileRename()</code> . Note that the renamed file remains in the <code>/PALM/Programs</code> directory; <code>VFSFileRename()</code> can't be used to move files from one directory to another.
	<pre>// volRefNum must have been previously defined; most likely, // it was returned by VFSVolumeEnumerate. err = VFSFileRename (volRefNum, "/PALM/Programs/foo.prc", "bar.prc"); if (err != 0) { // Handle error... }</pre> <hr/>
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	<u>VFSFileCreate()</u> , <u>VFSFileDelete()</u>

VFSFileResize Function

Purpose	Changes the size of an open file.
Declared In	VFSMgr.h
Prototype	SInt32 VFSFileResize (FileRef <i>fileRef</i> , UInt32 <i>ui32NewSize</i>)
Parameters	<p>→ <i>fileRef</i> File reference number passed back from VFSFileOpen().</p> <p>→ <i>ui32NewSize</i> The desired new size of the file. This can be larger or smaller than the current file size.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error codes:</p> <p>expErrNotOpen SYNCERR_COMM_NOT_INIT vfsErrFileBadRef vfsErrInvalidOperation vfsErrIsADirectory vfsErrNoFileSystem vfsErrVolumeFull</p> <p>For descriptions of all VFS Manager error codes, see “VFS Manager Error Codes” on page 645.</p>
Comments	<p>If the resizing of the file would make the current file pointer point beyond the end of the file, this function sets the file pointer to the end of the file.</p> <p>This function operates only on files and cannot be used with directories.</p>
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the C/C++ Sync Suite Companion).
See Also	VFSFileSize()

Virtual File System Manager API

VFSFileSeek

VFSFileSeek Function

Purpose Sets the position from which to read or write within an open file.

Declared In VFSMgr.h

Prototype SInt32 VFSFileSeek (FileRef *fileRef*,
 FileOrigin *origin*, SInt32 *offset*)

Parameters

→ *fileRef*

File reference number passed back from [VFSFileOpen\(\)](#).

→ *origin*

Origin to use when calculating the new position. The *offset* parameter indicates the desired new position relative to this origin, which must be one of the values defined in “[Seek Origins](#)” on page 576.

→ *offset*

Offset, either positive or negative, from the origin to which to set the current position. A value of zero positions you at the specified origin.

Returns If successful, returns 0.

If unsuccessful, returns one of the following error codes:

expErrNotOpen

SYNCERR_COMM_NOT_INIT

SYNCERR_REMOTE_BAD_ARG

vfsErrFileBadRef

vfsErrFileEOF

vfsErrInvalidOperation

vfsErrIsADirectory

vfsErrNoFileSystem

For descriptions of all VFS Manager error codes, see “[VFS Manager Error Codes](#)” on page 645.

Comments

If the resulting position of the file pointer would be beyond the end of the file, this function sets the position to the end of the file. Similarly, if the resulting position of the file pointer would be before the beginning of the file, this function sets the position to the beginning of the file.

This function operates only on files and cannot be used with directories.

Compatibility

VFS Manager version: All.

Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also

[VFSFileSize\(\)](#), [VFSFileTell\(\)](#)

Virtual File System Manager API

VFSFileSetAttributes

VFSFileSetAttributes Function

Purpose	Sets the attributes of an open file or directory.
Declared In	VFSMgr.h
Prototype	SInt32 VFSFileSetAttributes (FileRef <i>fileRef</i> , UInt32 <i>ui32Attributes</i>)
Parameters	<p>→ <i>fileRef</i> File reference number passed back from VFSFileOpen().</p> <p>→ <i>ui32Attributes</i> Attributes to associate with the file or directory. See “File and Directory Attributes” on page 572 for a list of values you can use when setting this parameter.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: expErrNotOpen SYNCERR_COMM_NOT_INIT SYNCERR_REMOTE_BAD_ARG vfsErrFileBadRef vfsErrInvalidOperation vfsErrNoFileSystem For descriptions of all VFS Manager error codes, see “ VFS Manager Error Codes ” on page 645.
Comments	You cannot use this function to set the vfsFileAttrDirectory or vfsFileAttrVolumeLabel attributes. The vfsFileAttrDirectory is set when you call VFSDirCreate() . The vfsFileAttrVolumeLabel attribute is set when you call VFSVolumeSetLabel() .
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	VFSFileGetAttributes() , VFSFileSetDate()

VFSFileSetDate Function

Purpose	Changes the dates of an open file or directory.
Declared In	VFSMgr.h
Prototype	SInt32 VFSFileSetDate (FileRef <i>fileRef</i> , UInt16 <i>whichDate</i> , UInt32 <i>ui32Date</i>)
Parameters	<p>→ <i>fileRef</i> File reference number passed back in VFSfileOpen().</p> <p>→ <i>whichDate</i> Specifies which date—creation, modification, or last access—to modify. Supply one of the constants defined in “Date Types” on page 570.</p> <p>Note that not all file systems are required to support all date types. If the supplied date type is not supported by the file system, VFSfileGetDate() returns <code>expErrUnsupportedOperation</code>.</p> <p>→ <i>ui32Date</i> The new date. Express this parameter in the standard Palm OS date format—the number of seconds since midnight (00:00:00) January 1, 1904.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error codes:</p> <p><code>expErrNotOpen</code> <code>expErrUnsupportedOperation</code> <code>SYNCERR_COMM_NOT_INIT</code> <code>SYNCERR_REMOTE_BAD_ARG</code> <code>vfsErrFileBadRef</code> <code>vfsErrFilePermissionDenied</code> <code>vfsErrInvalidOperation</code> <code>vfsErrNoFileSystem</code></p> <p>For descriptions of all VFS Manager error codes, see “VFS Manager Error Codes” on page 645.</p>
Comments	If the <i>whichDate</i> parameter is not one of the defined date type constants, this function returns <code>SYNCERR_REMOTE_BAD_ARG</code> .

Virtual File System Manager API

VFSFileSetDate

However, if *whichDate* is one of the defined constants but is not one supported by the file system, this function returns `expErrUnsupportedOperation`.

Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	VFSFileGetDate() , VFSFileSetAttributes()

VFSFileSize Function

Purpose	Gets the size of an open file.
Declared In	VFSMgr.h
Prototype	SInt32 VFSFileSize (FileRef <i>fileRef</i> , UIInt32 * <i>pui32FileSize</i>)
Parameters	<p>→ <i>fileRef</i> File reference number passed back from VFSFileOpen().</p> <p>← <i>pui32FileSize</i> Pointer to the size of the open file.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: expErrNotOpen SYNCERR_COMM_NOT_INIT vfsErrFileBadRef vfsErrInvalidOperation vfsErrIsADirectory vfsErrNoFileSystem For descriptions of all VFS Manager error codes, see “ VFS Manager Error Codes ” on page 645.
Comments	This function operates only on files and cannot be used with directories.
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	VFSFileResize() , VFSFileTell() , VFSVolumeSize()

Virtual File System Manager API

VFSFileTell

VFSFileTell Function

Purpose	Gets the current position of the file pointer within an open file.
Declared In	VFSMgr.h
Prototype	SInt32 VFSFileTell (FileRef <i>fileRef</i> , UInt32 * <i>pFilePos</i>)
Parameters	<p>→ <i>fileRef</i> File reference number passed back from VFSFileOpen().</p> <p>← <i>pFilePos</i> Pointer to the current position of the file pointer.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: expErrNotOpen SYNCERR_COMM_NOT_INIT vfsErrFileBadRef vfsErrInvalidOperation vfsErrIsADirectory vfsErrNoFileSystem For descriptions of all VFS Manager error codes, see “ VFS Manager Error Codes ” on page 645.
Comments	This function operates only on files and cannot be used with directories.
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the C/C++ Sync Suite Companion).
See Also	VFSFileSeek() , VFSFileSize()

VFSFileWrite Function

Purpose	Writes data to an open file.
Declared In	VFSMgr.h
Prototype	SInt32 VFSFileWrite (FileRef <i>fileRef</i> , UInt32 <i>numBytes</i> , const void * <i>pDataBuf</i> , UInt32 * <i>pNumBytesWritten</i>)
Parameters	<p>→ <i>fileRef</i> File reference number passed back from VFSFileOpen().</p> <p>→ <i>numBytes</i> The number of bytes to write.</p> <p>→ <i>pDataBuf</i> A pointer to a buffer containing the data to write.</p> <p>← <i>pNumBytesWritten</i> A 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.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error codes:</p> <p>expErrNotOpen SYNCERR_COMM_NOT_INIT SYNCERR_REMOTE_BAD_ARG vfsErrFileBadRef vfsErrFilePermissionDenied vfsErrInvalidOperation vfsErrIsADirectory vfsErrNoFileSystem vfsErrVolumeFull</p> <p>For descriptions of all VFS Manager error codes, see “VFS Manager Error Codes” on page 645.</p>
Comments	This function operates only on files and cannot be used with directories.

Virtual File System Manager API

VFSFileWrite

Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	<code>VFSExportDatabaseToFile()</code> , <code>VFSFileRead()</code>

VFSGetAPIVersion Function

Purpose	Retrieves the version of the Expansion Manager and VFS Manager APIs.
Declared In	<code>VFSMgr.h</code>
Prototype	<code>SInt32 VFSGetAPIVersion (UInt32 *pdwMajor, UInt32 *pdwMinor)</code>
Parameters	<p>$\leftarrow pdwMajor$ Pointer to variable for returning the <i>major</i> version number; pass NULL to ignore.</p> <p>$\leftarrow pdwMinor$ Pointer to variable for returning the <i>minor</i> version number; pass NULL to ignore.</p>
Returns	Always returns 0.
Comments	The Expansion Manager and VFS Manager APIs strive to maintain backward compatibility within a given <i>major</i> version number group. As new exported functions are added to the APIs or critical bugs are fixed, the <i>minor</i> version number of the APIs will be incremented and the documentation of the new functions will identify the APIs version number where they were first available. Conduits can check the version number using the <code>VFSGetAPIVersion()</code> function. For example, if a conduit requires a bug fix for a particular VFS Manager function that was made in VFS Manager API version number 2.1, the conduit must call <code>VFSGetAPIVersion()</code> to make sure that the <i>major</i> number is 2 and the <i>minor</i> number is 1 or greater before making calls to the new function.
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	“Versions of the VFS Manager API” on page 577

Virtual File System Manager API

VFSGetDefaultDirectory

VFSGetDefaultDirectory Function

Purpose	Retrieves the default directory on the given volume for files of a particular type.
Declared In	VFSMgr.h
Prototype	SInt32 VFSGetDefaultDirectory (UInt16 <i>volRefNum</i> , const char * <i>pszFileType</i> , char * <i>pszDirPath</i> , UInt16 * <i>pwPathLen</i>)
Parameters	<p>→ <i>volRefNum</i> Volume reference number passed back by VFSVolumeEnumerate().</p> <p>→ <i>pszFileType</i> A 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”. If you pass in a file extension, it must begin with a period ‘.’—for example “.prc”.</p> <p>← <i>pszDirPath</i> A pointer to the buffer that receives the default directory path for the requested file type. The caller must allocate this buffer before calling this function.</p> <p>↔ <i>pwPathLen</i> A pointer to the size of the path. On entry, set this to the size of <i>pszDirPath</i> buffer. On return, reflects the number of bytes copied to <i>pszDirPath</i>.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: SYNCERR_COMM_NOT_INIT SYNCERR_REMOTE_BAD_ARG vfsErrBadName vfsErrBufferOverflow vfsErrFileNotFoundException vfsErrInvalidOperation

For descriptions of all VFS Manager error codes, see “[VFS Manager Error Codes](#)” on page 645.

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.) If this function finds no match for either the specified media type for this volume or the requested file type, it returns `vfsErrFileNotFoundException`.

If a match is found, but the `pszDirPath` buffer is too small to hold the resulting path string, this function returns
`vfsErrBufferOverflow`.

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.

Example

This example illustrates how to use the `VFSGetDefaultDirectory()` function.

```
char fileTypeStr [] = ".prc";
char devicePathBuffer [MAX_PATH];
UInt16 bufLen = sizeof (devicePathBuffer);

SInt32 retval = VFSGetDefaultDirectory
    (volRefNum, fileTypeStr, devicePathBuffer, &bufLen);
if (0 == retval)
{
    // Got the default directory.
    // Perform further operations here.
}
else
{
    // Could not get the default path on the card:
    // process error codes.
}
```

Compatibility

VFS Manager version: All.

Palm OS version: 4.0 or later. Implemented only if the VFS Manager

Virtual File System Manager API

VFSGetDefaultDirectory

is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also [VFSDirEntryEnumerate\(\)](#)

VFSImportDatabaseFromFile Function

Purpose Creates a database from the specified PDB or PRC file on an expansion card. Works only with classic databases.

Declared In VFSMgr.h

Prototype

```
SInt32 VFSImportDatabaseFromFile  
    (UInt16 volRefNum, const char *pszPathName,  
     UInt16 *pwCardNumber, LocalID *pDbId)
```

Parameters

→ *volRefNum*
Volume reference number (passed back by [VFSVolumeEnumerate\(\)](#)) of the volume from which to get the source file.

→ *pszPathName*
A pointer to the full path and name of the source file to get.

← *pwCardNumber*
A pointer to a variable that receives the [memory card](#) number of the newly-created database. If the database already resides in the storage heap, the card number of the existing database is passed back and the error SYNCERR_FILE_ALREADY_EXIST is returned.

← *pDbId*
A 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 passed back and the error SYNCERR_FILE_ALREADY_EXIST is returned.

Returns If successful, returns 0.

If unsuccessful, returns one of the following error codes:

SYNCERR_COMM_NOT_INIT

SYNCERR_FILE_ALREADY_EXIST

SYNCERR_REMOTE_BAD_ARG

vfsErrBadName

vfsErrInvalidOperation

For descriptions of all VFS Manager error codes, see “[VFS Manager Error Codes](#)” on page 645.

Virtual File System Manager API

VFSImportDatabaseFromFile

Comments This function imports a PDB or PRC file on an expansion card into a new database in the handheld storage heap. If the database already exists, this function passes back values in *pwCardNumber* and *pDbID* for the existing database and returns an error code of SYNCERR_FILE_ALREADY_EXIST.

This function is used, for example, to copy applications from a storage card to main memory.

IMPORTANT: With VFS Manager API version 1.1, this function is deprecated for use with handhelds running Palm OS Cobalt; call [VFSImportDatabaseFromFileEx\(\)](#) instead. The reason for replacing this function is that memory card numbers are not supported by Palm OS Cobalt and local IDs are insufficient to identify schema databases and are cumbersome to identify extended and classic databases.

However, this function is still supported for use with Palm OS 4.x and Palm OS Garnet.

Example This example illustrates the use of the `VFSImportDatabaseFromFile()` function.

```
SInt32 retval = VFSImportDatabaseFromFile (volRefNum,  
    "/Palm/Launcher/Contacts.pdb", &cardNo, &dBid);
```

Compatibility VFS Manager version: All.
Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see "[Checking for Expansion Cards](#)" on page 59 in the *C/C++ Sync Suite Companion*).

See Also [VFSExportDatabaseToFile\(\)](#), [VFSFileRead\(\)](#)

VFSImportDatabaseFromFileEx Function

Purpose	Creates a database from the specified image file on an expansion card. Works with schema, extended, and classic databases only on handhelds running Palm OS Cobalt, version 6.0.1 or later.
Declared In	<code>VFSMgr.h</code>
Prototype	<pre>SInt32 VFSImportDatabaseFromFileEx (UInt16 volRefNum, char *pszPathName, DBDatabaseInfo *pDbInfo)</pre>
Parameters	<p>→ <i>volRefNum</i> Volume reference number (passed back by VFSVolumeEnumerate()) of the volume from which to get the source file.</p> <p>→ <i>pszPathName</i> A pointer to the full path and name of the source file to get. Do not specify NULL.</p> <p>← <i>pDbInfo</i> A pointer to a DBDatabaseInfo structure. If successful, this structure contains values in all section 1 fields, which describe the database that this function created.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error codes:</p> <p style="margin-left: 40px;">SYNCERR_COMM_NOT_INIT SYNCERR_FILE_ALREADY_EXIST SYNCERR_REMOTE_BAD_ARG SYNCERR_UNKNOWN_REQUEST The handheld is running a version of Palm OS that this function does not support. See “Compatibility” below.</p> <p style="margin-left: 40px;">vfsErrBadName vfsErrInvalidOperation</p> <p>For descriptions of all VFS Manager error codes, see “VFS Manager Error Codes” on page 645.</p>

Virtual File System Manager API

VFSImportDatabaseFromFileEx

Comments This function imports an image file on an expansion card into a new database in the handheld storage heap. The file is an image of either a schema, extended, or classic database.

This function does the opposite of [VFSExportDatabaseToFileEx\(\)](#). Use it, for example, to copy applications or databases from an expansion card to primary storage on the handheld.

This function is a replacement for [VFSImportDatabaseFromFile\(\)](#), which identifies databases only by memory card number and local ID. The reason for replacing this function is that memory card numbers are not supported by Palm OS Cobalt and local IDs are insufficient to identify schema databases and somewhat cumbersome to identify extended and classic databases.

Example This example illustrates the use of the `VFSImportDatabaseFromFileEx()` function.

```
UInt16 volRefNum; // Get this via VFSVolumeEnumerate().  
DBDatabaseInfo dbInfo;  
  
SInt32 retval = VFSImportDatabaseFromFileEx (volRefNum,  
    "/Palm/Launcher/MySchemaDatabase.sdb", &dbInfo);
```

Compatibility VFS Manager version: 1.1 or later.
Palm OS version: 6.0.1 or later. Implemented only if the VFS Manager is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also [DBDatabaseInfo](#), [VFSExportDatabaseToFileEx\(\)](#)

VFSSupport Function

Purpose	Determines whether the Expansion Manager and VFS Manager are present on the handheld, their version if present, and gets expansion slot and volume information.
Declared In	<code>VFSMgr.h</code>
Prototype	<code>SInt32 VFSSupport (UInt32 &dwExpansionMgrVersion, UInt32 &dwVolumesAvailable)</code>
Parameters	<p>$\leftarrow dwExpansionMgrVersion$ When this parameter passes back a zero value, no expansion slot is present. A nonzero value is the version of Expansion Manager on the handheld. The VFS Manager version number is identical.</p> <p>$\leftarrow dwVolumesAvailable$ When this parameter passes back a zero value, either no file system is present on the card in the slot or no card is in the slot. A nonzero value is the number of volumes present on the card.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error codes:</p> <p><code>SYNCERR_BAD_ARG</code></p> <p>For descriptions of all VFS Manager error codes, see “VFS Manager Error Codes” on page 645.</p>
Comments	This information has already been obtained by the desktop VFS Manager, so no additional calls are made to the handheld at the time you call this function. If either pointer is <code>NULL</code> , this function returns <code>SYNCERR_BAD_ARG</code> .
Compatibility	VFS Manager version: All. Palm OS version: 4.0.
See Also	VFSGetAPIVersion() , “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>

Virtual File System Manager API

VFSVolumeEnumerate

VFSVolumeEnumerate Function

Purpose	Enumerates the mounted volumes and retrieves a list of volume reference numbers.
Declared In	VFSMgr.h
Prototype	SInt32 VFSVolumeEnumerate (UInt16 * <i>pwNumVolumes</i> , UInt16 * <i>pwVolRefList</i>)
Parameters	<i>pwNumVolumes</i> A pointer to the number of volumes successfully enumerated. <i>pwVolRefList</i> On exit, a pointer to an array of volume reference numbers. If the caller passes in NULL, the function passes back no volume reference numbers. If NULL is not passed in, the caller must allocate sufficient space to hold all volume reference numbers.
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: <code>expErrEnumerationEmpty</code> <code>SYNCERR_COMM_NOT_INIT</code> <code>vfsErrInvalidOperation</code> For descriptions of all VFS Manager error codes, see “ VFS Manager Error Codes ” on page 645.
Comments	This function passes back a list of reference numbers of all of the volumes that are mounted. The list can span across expansion cards, if multiple cards are present. To find which card and slot this volume is mounted from, call VFSVolumeInfo() . Before calling VFSVolumeEnumerate() to get volume reference numbers, the caller must allocate enough space for the array pointed to by <i>pwVolRefList</i> . If the caller passes in NULL for <i>pwVolRefList</i> , the function returns only the number of volumes. Use this to allocate the array and call VFSVolumeEnumerate() again to get the volume reference numbers.

Example The following example shows how to use `VFSVolumeEnumerate()` to get the number of mounted volumes, allocate a buffer, and get the list of volume reference numbers.

```
UInt16 numVolumes = 0;
UInt16 *pwVolRefNumList;

// The first call returns only the number of mounted volumes,
// not their reference numbers.
VFSVolumeEnumerate (&numVolumes, NULL);
if (numVolumes)
{
    // Allocate buffer for volume reference list.
    pwVolRefNumList = new WORD [numVolumes];
    if (pwVolRefNumList != NULL)
    {
        // Get the volume reference numbers.
        VFSVolumeEnumerate (&numVolumes, pwVolRefNumList);
    }
}
```

Compatibility VFS Manager version: All.
Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also [VFSVolumeInfo\(\)](#)

Virtual File System Manager API

VFSVolumeFormat

VFSVolumeFormat Function

Purpose	Formats and mounts the first volume installed in a given slot.
Declared In	VFSMgr.h
Prototype	<pre>SInt32 VFSVolumeFormat (HSByte <i>byMountFlags</i>, UInt16 <i>fsLibRefNum</i>, VFSAnyMountParamPtr <i>pVfsMountParam</i>, UInt16 <i>vfsMountParamLen</i>)</pre>
Parameters	<p>→ <i>byMountFlags</i> Flags that control how the volume should be formatted. Currently, the only value defined in “Volume Format/Mount Flags” on page 580 that is not reserved is <code>vfsMountFlagsUseThisFileSystem</code>. Pass this flag to cause the volume to be formatted using the file system specified by <code>fsLibRefNum</code>. Pass zero (0) to have the VFS Manager attempt to format the volume using a file system appropriate to the slot.</p> <p>→ <i>fsLibRefNum</i> Reference number of the file system library for which the volume should be formatted. If <i>byMountFlags</i> is not set to <code>vfsMountFlagsUseThisFileSystem</code>, this parameter is ignored.</p> <p>↔ <i>pVfsMountParam</i> Parameters to be used when mounting the volume after it has been formatted. Supply a pointer to a <code>VFSAnyMountParamType</code> structure. Note that you must pass in a pointer to a different structure type depending on the value of <i>pVfsMountParam->mountClass</i>. For example, if <i>mountClass</i> is set to <code>vfsMountClass_SlotDriver</code>, then the <i>pVfsMountParam</i> you pass in must point to a <code>VFSSlotMountParamType</code> structure. Upon exit, this points to a structure of the same type containing a new volume reference number.</p> <p>→ <i>vfsMountParamLen</i> The length in bytes of the structure passed via <i>pVfsMountParam</i>.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: <code>expErrNotEnoughPower</code>

expErrNotOpen
SYNCERR_COMM_NOT_INIT
vfsErrInvalidOperation
vfsErrNoFileSystem

For descriptions of all VFS Manager error codes, see “[VFS Manager Error Codes](#)” on page 645.

Comments

`VFSVolumeFormat()` attempts to find a compatible file system library on the handheld to mount the volume. A volume can be either mounted or unmounted at the time you call this function. `VFSVolumeFormat()` also remounts the volume if the format succeeds. (The handheld slot driver provided by PalmSource, Inc. currently supports only one volume per slot.)

NOTE: For a card that has not been previously formatted (and therefore does not have a volume), set the `volRefNum` member of the [`VFSAnyMountParamType`](#) structure to `vfsInvalidVolRef`. Upon exit, this function passes back a valid volume reference number in a structure of the same type.

To use `VFSVolumeFormat()` with a file system based on a slot driver:

- The `pVfsMountParam` parameter must point to a `VFSSlotMountParamType` structure.
- The `pVfsMountParam->mountClass` member must be set to `vfsMountClass_SlotDriver`.
- The `pVfsMountParam->slotLibRefNum` member may be set to `vfsInvalidSlotLibRefNum` (which causes the handheld HotSync® client to look up the proper driver) or to the value obtained for the [`VolumeInfoType`](#) structure by calling [`VFSVolumeInfo\(\)`](#).

If `vfsMountFlagsUseThisFileSystem` is passed as a `byMountFlags` value, `VFSVolumeFormat()` attempts to format the volume using the file system library specified by `fsLibRefNum`. Typically the `byMountFlags` parameter is not set. In this case `VFSVolumeFormat()` tries to find a compatible library to format the volume.

Virtual File System Manager API

VFSVolumeFormat

Example The following code excerpt formats a volume on a physical slot using a compatible file system.

```
VFSSlotMountParamType stSlotMountParam;

stSlotMountParam.vfsMountParam.volRefNum = volRefNum; // Or vfsInvalidVolRef if
                                                       // the card has not been
                                                       // formatted.
stSlotMountParam.vfsMountParam.reserved = 0;
stSlotMountParam.vfsMountParam.mountClass = vfsMountClass_SlotDriver;
stSlotMountParam.slotLibRefNum = vfsInvalidSlotLibRefNum;
stSlotMountParam.slotRefNum = slotRefNum;
// We get this from ExpSlotEnumerate().

SInt32 retval = VFSVolumeFormat (0, 0, &stSlotMountParam,
                           sizeof(stSlotMountParam));
```

Compatibility VFS Manager version: All.
Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also [VFSVolumeInfo\(\)](#)

VFSVolumeGetLabel Function

Purpose	Gets the volume label for a particular volume.
Declared In	VFSMgr.h
Prototype	SInt32 VFSVolumeGetLabel (UInt16 <i>volRefNum</i> , char * <i>pszVolLabel</i> , UInt16 * <i>pwBufLen</i>)
Parameters	<p>→ <i>volRefNum</i> Volume reference number (passed back by VFSVolumeEnumerate()) of the volume for which to get the label.</p> <p>← <i>pszVolLabel</i> A pointer to a character buffer that receives the volume name upon return.</p> <p>→ <i>pwBufLen</i> A pointer to the length, in bytes, of the <i>pszVolLabel</i> buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error codes:</p> <p>expErrNotOpen SYNCERR_COMM_NOT_INIT SYNCERR_REMOTE_BAD_ARG vfsErrBufferOverflow vfsErrInvalidOperation vfsErrNameShortened vfsErrNoFileSystem vfsErrVolumeBadRef</p> <p>For descriptions of all VFS Manager error codes, see “VFS Manager Error Codes” on page 645.</p>
Comments	Volume reference numbers can change each time the handheld mounts 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 lowercase characters, in any character set as well as the following special characters:

\$ % ' - _ @ ~ ` ! () ^ # & + , ; = [].

Virtual File System Manager API

VFSVolumeGetLabel

Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	<u>VFSVolumeSetLabel()</u>

VFSVolumeInfo Function

Purpose	Gets information about the specified volume.
Declared In	<code>VFSMgr.h</code>
Prototype	<code>SInt32 VFSVolumeInfo (UInt16 volRefNum, VolumeInfoType *pVolInfo)</code>
Parameters	<p>→ <i>volRefNum</i> Volume reference number (passed back by VFSVolumeEnumerate()) of the volume for which to get information.</p> <p>← <i>pVolInfo</i> 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.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error codes:</p> <p><code>expErrNotOpen</code> <code>SYNCERR_COMM_NOT_INIT</code> <code>vfsErrInvalidOperation</code> <code>vfsErrNoFileSystem</code> <code>vfsErrVolumeBadRef</code></p> <p>For descriptions of all VFS Manager error codes, see “VFS Manager Error Codes” on page 645.</p>
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	VFSVolumeGetLabel() , VFSVolumeSize()

Virtual File System Manager API

VFSVolumeSetLabel

VFSVolumeSetLabel Function

Purpose	Changes the volume label for a mounted volume.
Declared In	<code>VFSMngr.h</code>
Prototype	<code>SInt32 VFSVolumeSetLabel (UInt16 volRefNum, const char *pszVolLabel)</code>
Parameters	<p>→ <i>volRefNum</i> Volume reference number (passed back by VFSVolumeEnumerate()) of the volume for which to set the label.</p> <p>→ <i>pszVolLabel</i> Pointer to the label to apply to the specified volume. This string must be NULL-terminated.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: <code>expErrNotOpen</code> <code>SYNCERR_COMM_NOT_INIT</code> <code>SYNCERR_REMOTE_BAD_ARG</code> <code>vfsErrBadName</code> <code>vfsErrInvalidOperation</code> <code>vfsErrNameShortened</code> <code>vfsErrNoFileSystem</code> <code>vfsErrVolumeBadRef</code> For descriptions of all VFS Manager error codes, see “ VFS Manager Error Codes ” on page 645.
Comments	Volume labels can be up to 255 characters long. They can contain any normal character, including spaces and lowercase characters, in any character set as well as the following special characters: \$ % ' - _ @ ~ ` ! () ^ # & + , ; = []. See “ Naming Volumes ” on page 69 in the <i>C/C++ Sync Suite Companion</i> for guidelines on naming.

NOTE: Most conduits or applications should not need to call `VFSVolumeSetLabel()`. This function may create or delete a file in the root directory, which would invalidate any current calls to [VFSDirEntryEnumerate\(\)](#).

Compatibility

VFS Manager version: All.

Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “[Checking for Expansion Cards](#)” on page 59 in the *C/C++ Sync Suite Companion*).

See Also

[VFSVolumeGetLabel\(\)](#)

Virtual File System Manager API

VFSVolumeSize

VFSVolumeSize Function

Purpose	Determines the total amount of space on a volume, as well as the amount that is currently being used.
Declared In	VFSMgr.h
Prototype	SInt32 VFSVolumeSize (UInt16 <i>volRefNum</i> , UInt32 * <i>pui32SizeUsed</i> , UInt32 * <i>pui32TotalCapacity</i>)
Parameters	<p>→ <i>volRefNum</i> Volume reference number (passed back by VFSVolumeEnumerate()) of the volume for which to get the size.</p> <p>← <i>pui32SizeUsed</i> A pointer to a variable that receives the amount of space, in bytes, in use on the volume.</p> <p>← <i>pui32TotalCapacity</i> A pointer to a variable that receives the total capacity of the volume, in bytes.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error codes: expErrNotOpen SYNCERR_COMM_NOT_INIT vfsErrInvalidOperation vfsErrNoFileSystem vfsErrVolumeBadRef For descriptions of all VFS Manager error codes, see “ VFS Manager Error Codes ” on page 645.
Compatibility	VFS Manager version: All. Palm OS version: 4.0 or later. Implemented only if the VFS Manager is present on the handheld (see “ Checking for Expansion Cards ” on page 59 in the <i>C/C++ Sync Suite Companion</i>).
See Also	VFSVolumeInfo()

VFS Manager Error Codes

[Table 10.1](#) lists the values of error codes that the VFS Manager API functions can return. The description of each function states which errors each function can return.

All of the named error codes below are defined as preprocessor constants, which are declared in the `VFSERR.h` header file.

Expansion Manager error codes (`exp...`) and Sync Manager error codes (`SYNC...`) can also be returned (see “[Expansion Manager API Error Codes](#)” on page 560 and “[Common Sync Manager Error Codes](#)” on page 486).

Table 10.1 VFS Manager error codes

Value	Code	Description
0x00002A01L	<code>vfsErrBufferOverflow</code>	The supplied buffer is too small.
0x00002A02L	<code>vfsErrFileGeneric</code>	Generic file error.
0x00002A03L	<code>vfsErrFileBadRef</code>	The file reference number is invalid: it has been closed or was not obtained from VFSFileOpen() .
0x00002A04L	<code>vfsErrFileStillOpen</code>	The file is still open—for example, trying to delete an open file.
0x00002A05L	<code>vfsErrFilePermissionDenied</code>	Permission denied to perform requested operation—for example, an attempt to write to a read-only file or to read a file already opened in the <code>vfsModeExclusive</code> mode.
0x00002A06L	<code>vfsErrFileAlreadyExists</code>	A file or a directory with this name exists in this location already.

Virtual File System Manager API

VFS Manager Error Codes

Table 10.1 VFS Manager error codes (*continued*)

Value	Code	Description
0x00002A07L	vfsErrFileEOF	The file pointer has been moved to the end of the file. This code is not considered an error.
0x00002A08L	vfsErrFileNotFoundException	The file was not found in the specified path.
0x00002A09L	vfsErrVolumeBadRef	The volume reference number is invalid.
0x00002A0AL	vfsErrVolumeStillMounted	The volume is still mounted.
0x00002A0BL	vfsErrNoFileSystem	None of the file systems installed on the handheld support this operation.
0x00002A0CL	vfsErrBadData	The operation could not be completed because of invalid data—for example, importing a database from a corrupted .prc file.
0x00002A0DL	vfsErrDirNotEmpty	The directory is not empty and therefore cannot be deleted.
0x00002A0EL	vfsErrBadName	Invalid filename, path, or volume label. See “ Naming Files ” on page 72, “ Directory Paths ” on page 74, or “ Naming Volumes ” on page 69 in the C/C++ Sync Suite Companion.
0x00002A0FL	vfsErrVolumeFull	There is insufficient space left on the volume.
0x00002A10L	vfsErrUnimplemented	This call is not implemented.
0x00002A11L	vfsErrNotADirectory	This operation can be performed only on a directory.

Table 10.1 VFS Manager error codes (*continued*)

Value	Code	Description
0x00002A12L	vfsErrIsADirectory	This operation can be performed only on a regular file, not a directory.
0x00002A13L	vfsErrDirectoryNotFound	The path, excluding filename, does not exist or no default directory is registered for this file type.
0x00002A14L	vfsErrNameShortened	A volume name or filename was automatically shortened to conform to the file system specification.
0x00002A30L	vfsErrInvalidOperation	A file system is not present or the VFS Manager function is not valid.
0x00002A31L	vfsErrDiskFileAccess	Failed to create or open the disk file on the desktop. Use Windows' <code>GetLastError()</code> function for details.
0x00002A32L	vfsErrDiskFull	Not enough space on the desktop's disk.
0x00002A33L	vfsErr FileAccessOther	Generic desktop file access error. Use Windows' <code>GetLastError()</code> function for details. If returned by <u>VFSFileGet()</u> , could not access or map the desktop file—for example, because of insufficient memory on the desktop.

Virtual File System Manager API

VFS Manager Error Codes



Part III

Desktop Support

APIs

This part describes the APIs that desktop applications and installers can call, usually not during a HotSync operation. However, some of these can be called by a conduit during a HotSync operation.

<u>Conduit Manager API</u>	651
<u>Install Conduit Manager API</u>	847
<u>Notifier Install Manager API</u>	909
<u>HotSync Manager API</u>	927
<u>Install Aide API</u>	951
<u>User Data API</u>	1009
<u>Password Library</u>	1093
<u>Desktop Application Notifier API</u>	1097

Conduit Manager API

The Conduit Manager provides an API for registering and unregistering a synchronization conduit with HotSync® Manager. It also provides utility functions for accessing the conduit configuration entries that Conduit Manager creates when you register a conduit. For a definition of each configuration entry, see [Appendix A, “Configuration Entries,”](#) on page 175 in the *Introduction to Conduit Development*.

The Conduit Manager functions are available in CondMgr.dll and declared in CondMgr.h.

The sections in this chapter are:

Conduit Manager Structures	652
Conduit Manager Constants	660
Conduit Manager Functions	665
Conduit Manager Error Codes	843

For information on using the Conduit Manager, see [Chapter 9, “Writing an Installer,”](#) on page 99 in the *C/C++ Sync Suite Companion*.

Conduit Manager Structures

This section describes the following data structures that you use with the Conduit Manager API.

<u>CmConduitType</u>	Specifies conduit registration information that you pass into <u>CmInstallConduit()</u> to register a conduit.
<u>CmConduitType2</u>	Defines conduit registration information that is either passed in when you register a conduit or is passed back by Conduit Manager when you request information about a conduit.
<u>CmDiscoveryInfoType</u>	Receives information about how HotSync Manager discovers the conduit.
<u>CM_CREATORLIST_ITEM_TYPE</u>	Receives the creator ID associated with a conduit.

CmConduitType Struct

Purpose Specifies conduit registration information that you pass into [CmInstallConduit\(\)](#) to register a conduit.

Declared In CondMgr.h

Prototype

```
typedef struct {
    int iStructureVersion;
    int iStructureSize;
    int iType;
    char szCreatorID[CM_CREATOR_ID_SIZE];
    DWORD dwPriority;
    int iConduitNameOffset;
    int iDirectoryOffset;
    int iFileOffset;
    int iRemoteDBOffset;
    int iUsernameOffset;
    int iTitleOffset;
    int iInfoOffset;
} CmConduitType
```

Fields **iStructureVersion**

Specifies the version of this data structure that the caller is using, which is defined as the constant CONDUIT_VERSION in “[Miscellaneous Constants](#)” on page 664. The Conduit Manager uses this to determine how to process information in the structure.

iStructureSize

Specifies the size of the structure, in bytes.

iType

Specifies the conduit information type, as described in “[Conduit Information Types](#)” on page 661.

szCreatorID

Specifies the conduit’s creator ID set in [Creator](#).

dwPriority

Specifies the conduit’s [Priority](#).

iConduitNameOffset

Specifies the offset from the beginning of this structure to the first character of the conduit name string set in [Conduit](#).

Conduit Manager API

CmConduitType

iDirectoryOffset

Specifies the offset from the beginning of this structure to the first character of the directory name string set in [Directory](#).

iFileOffset

Specifies the offset from the beginning of this structure to the first character of the filename string set in [File](#).

iRemoteDBOffset

Specifies the offset from the beginning of this structure to the first character of the handheld database name string set in [Remote](#).

iUsernameOffset

Specifies the offset from the beginning of this structure to the first character of the user name string set in [Username](#).

iTitleOffset

Specifies the offset from the beginning of this structure to the first character of the title string set in [Name](#).

iInfoOffset

Specifies the offset from the beginning of this structure to the first character of the information string set in [Information](#).

Comments

The *CmConduitType* structure includes several strings, each of which is stored after the end of the structure fields. The structure includes an offset field for each string; each offset specifies the number of bytes from the beginning of the structure to the beginning of that string.

NOTE: The *CmConduitType* structure includes fields for several conduit configuration entries that are deprecated in CDK versions 6.0 and later. PalmSource recommends that you use the [*CmConduitType2*](#) structure with functions like [*CmInstallConduitByStruct\(\)*](#) or [*CmInstallSystemConduitByStruct\(\)*](#) instead.

For more information on registering a conduit using this structure, see “[Writing a Single Structure](#)” on page 106 in the *C/C++ Sync Suite Companion*.

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [CmInstallConduit\(\)](#)

Conduit Manager API

CmConduitType2

CmConduitType2 Struct

Purpose Defines conduit registration information that is either passed in when you register a conduit or is passed back by Conduit Manager when you request information about a conduit.

Declared In CondMgr.h

Prototype

```
typedef struct {
    DWORD dwCreatorID;
    DWORD dwPriority;
    TCHAR szConduitPath[255];
    TCHAR szLocalDirectory[255];
    TCHAR szLocalFile[255];
    TCHAR szRemoteDB[32];
    TCHAR szTitle[255];
} CmConduitType2
```

Fields

dwCreatorID
Defines the conduit's creator ID. This value is set in the conduit's [Creator](#) configuration entry.

dwPriority
Defines the priority with which HotSync Manager runs the conduit. This value is set in the conduit's [Priority](#) configuration entry.

szConduitPath
Defines the full path and filename (or only filename) of the conduit. This value is set in the conduit's [Conduit](#) configuration entry.

szLocalDirectory
Defines the conduit's directory name. This value is set in the conduit's [Directory](#) configuration entry.

szLocalFile
Defines the desktop file that the conduit synchronizes with. This value is set in the conduit's [File](#) configuration entry.

szRemoteDB
Defines the name of a database on the handheld that the conduit synchronizes with. Note that a conduit can read and write any number of databases, so defining this value is optional. This value is set in the conduit's [Remote](#) configuration entry.

szTitle

Defines the display name of the conduit. This value is set in the [Name](#) configuration entry.

Comments

This structure is used by the APIs introduced in Conduit Manager versions 3 and later to access information about a conduit. The functions that use this structure are listed under “See Also” below. Each field of this structure corresponds to a conduit configuration entry (see [Table A.1](#) on page 177).

This structure has two advantages over the [CmConduitType](#) structure. The [CmConduitType2](#) structure:

- has a fixed size, which makes it easier to pass as a function parameter
- includes only the required conduit configuration entries, plus several optional ones that are used by most conduits

Compatibility

Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also

[CmGetConduitByIndex\(\)](#),
[CmGetSystemConduitByCreator\(\)](#),
[CmGetSystemConduitByIndex\(\)](#),
[CmInstallConduitByStruct\(\)](#),
[CmInstallSystemConduitByStruct\(\)](#),
[FmGetCurrentUserConduitByIndex\(\)](#),
[FmGetSystemConduitByIndex\(\)](#)

CmDiscoveryInfoType Struct

Purpose Receives information about how HotSync Manager discovers the conduit.

Declared In CondMgr.h

Prototype

```
typedef struct {
    BOOL bDiscoverableViaFolder;
    BOOL bDiscoverableViaRegistry;
    BOOL bLoadable;
} CmDiscoveryInfoType
```

Fields

bDiscoverableViaFolder
Receives TRUE if the conduit is folder-registered—that is, if Conduit Manager discovers the conduit by its presence in the system or user's Conduits folder.

bDiscoverableViaRegistry
Receives TRUE if the conduit is conventionally registered—that is, if HotSync Manager discovers the conduit by looking in the conduit configuration entries.

bLoadable

Receives TRUE if HotSync Manager can successfully load the conduit.

Comments The boolean values in fields `bDiscoverableViaFolder` and `bDiscoverableViaRegistry` are mutually exclusive.

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [CmGetDiscoveryInfoByIndex\(\)](#),
[CmGetSystemDiscoveryInfoByIndex\(\)](#)

CM_CREATORLIST_ITEM_TYPE Struct

Purpose	Receives the creator ID associated with a conduit.
Declared In	CondMgr.h
Prototype	<pre>typedef struct { char szCreatorID[CM_CREATOR_ID_SIZE]; int iReserved; } CM_CREATORLIST_ITEM_TYPE typedef CM_CREATORLIST_ITEM_TYPE *CM_CREATORLIST_TYPE</pre>
Fields	szCreatorID Receives the creator ID associated with a conduit. iReserved Receives an integer value, which is reserved for future use.
Comments	CM_CREATORLIST_TYPE is defined as a pointer to an array of CM_CREATORLIST_ITEM_TYPE structures that is passed back by CmGetCreatorIDList() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	CmGetCreatorIDList()

Conduit Manager Constants

This section describes the following groups of preprocessor constants that you use with the Conduit Manager API.

<u>Conduit Information Types</u>	Define the type of conduit, which in turn defines the location of information used by the conduit.
<u>Communications Ports</u>	Indicates the type of connection for which to get or set the COM port.
<u>Conduit Manager Versions</u>	Defines the Conduit Manager version numbers that <u>CmGetLibVersion()</u> can return.
<u>Miscellaneous Constants</u>	Define miscellaneous constants for use with the Conduit Manager.

Conduit Information Types

Purpose	Define the type of conduit, which in turn defines the location of information used by the conduit.
Declared In	<code>CondMgr.h</code>
Constants	<pre>#define CONDUIT_APPLICATION 1 Indicates that the conduit information is stored in the application information location. #define CONDUIT_COMPONENT 0 Indicates that the conduit information is stored in the component information location. #define CONDUIT_CONDUITS 10 For future use. Indicates that the conduit information is stored in the conduits information location.</pre>
Comments	Specify one of these values when calling CmInstallCreator() or CmInstallConduit() . For almost all conduits, specify CONDUIT_APPLICATION. PalmSource recommends that you use CmInstallConduitByStruct() instead, which does not require that you specify one of these values.
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	CmInstallCreator() , CmInstallConduit()

Communications Ports

Purpose	Indicates the type of connection for which to get or set the COM port.
Declared In	CondMgr.h
Constants	<pre>#define DIRECT_COM_PORT 0 Specifies the DirectComPort entry in HotSync Manager's configuration entries. #define MODEM_COM_PORT 1 Specifies the ModemComPort entry in HotSync Manager's configuration entries. #define MAX_COM_PORT MODEM_COM_PORT Specifies the maximum range of communications port constants.</pre>
Comments	<p>Pass CmGetComPort () (or CmSetComPort ()) one of these values to indicate whether to retrieve (or to set) the name of the port used for a direct (local) or modem connection. These values specify which HotSync Manager configuration entries to read or write.</p> <p>Note that these values are set for the entire system, not per Windows user.</p>
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	CmGetComPort () , CmSetComPort ()

Conduit Manager Versions

Purpose	Defines the Conduit Manager version numbers that CmGetLibVersion() can return.
Declared In	CondMgr.h
Constants	<pre>#define CM_INITIAL_LIB_VERSION 0x0001 The value returned by Conduit Manager version 1. #define CM_UPDATE_1 0x0002 The value returned by Conduit Manager version 2. #define CM_UPDATE_2 0x0003 The value returned by Conduit Manager version 3. This version ships with HotSync Manager versions 6.0 and later.</pre>
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	CmGetLibVersion()

Conduit Manager API

Miscellaneous Constants

Miscellaneous Constants

Purpose Define miscellaneous constants for use with the Conduit Manager.

Declared In CondMgr.h

Constants

```
#define CM_CREATOR_ID_SIZE 8
    The size, in chars, of a creator ID. This is used in some data
    structures to define the size of a character array that stores a
    creator ID.

#define CONDUIT_VERSION 100
    The current version of the Conduit Manager's
    CmConduitType structure. This is not the version of
    Conduit Manager that is returned by CmGetLibVersion\(\).

#define CM_CONDUIT_BUFFER_OFFSET
    sizeof(CmConduitType)
    Defines an offset from the beginning of a CmConduitType
    structure.

#define CM_MIN_CONDUITTYPE_SIZE
    sizeof(CmConduitType)
    Defines the minimum size of a CmConduitType structure.

#define FILEINSTALLTYPE_SIZE
    sizeof(FileInstallType)
    Defines the size of the FileInstallType structure.
```

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

Conduit Manager Functions

This section describes the following functions, which allow you to register conduits and to retrieve information from the [configuration entries](#).

[CmConvertCreatorIDToString\(\)](#)

Converts a conduit creator ID from a DWORD to a string.

[CmConvertStringToCreatorID\(\)](#)

Converts a conduit creator ID from a string to a DWORD.

[CmGetBackupConduit\(\)](#)

Retrieves the path or filename of the backup conduit for the current Windows user.

[CmGetComPort\(\)](#)

Retrieves the name of the COM port that HotSync Manager uses for the specified type of connection on the desktop computer.

[CmGetConduitByCreator\(\)](#)

Retrieves information about a conduit registered for the current Windows user, given the conduit's creator ID.

[CmGetConduitByIndex\(\)](#)

Retrieves information about a conduit registered for the current Windows user, given the conduit's index.

[CmGetConduitCount\(\)](#)

Returns the number of conduits that are registered with HotSync Manager for the current Windows user.

[CmGetConduitCreatorID\(\)](#)

Retrieves the creator ID of a conduit that is registered for the current Windows user.

Conduit Manager API

Conduit Manager Functions

[CmGetCorePath\(\)](#)

Retrieves the path of the HotSync users' directory for the current Windows user.

[CmGetCreatorArgument\(\)](#)

(*Deprecated*) Retrieves the value of the Arguments configuration entry for the specified conduit.

[CmGetCreatorDirectory\(\)](#)

Retrieves the name of the directory in which a user-registered conduit stores its user data.

[CmGetCreatorFile\(\)](#)

Retrieves the name of the data file for a user-registered conduit.

[CmGetCreatorIDList\(\)](#)

Retrieves a list of the creator IDs of all conduits registered for the current Windows user.

[CmGetCreatorInfo\(\)](#)

(*Deprecated*) Retrieves a string associated with a user-registered conduit.

[CmGetCreatorIntegrate\(\)](#)

(*Deprecated*) Retrieves the value of the Integrate configuration entry for the specified conduit.

[CmGetCreatorModule\(\)](#)

(*Deprecated*) Retrieves the value of the Module configuration entry for the specified conduit.

[CmGetCreatorName\(\)](#)

Retrieves the filename of a user-registered conduit.

[CmGetCreatorPriority\(\)](#)

Retrieves the execution priority for a user-registered conduit.

<u>CmGetCreatorRemote()</u>	Retrieves the name of a database on the handheld that is associated with a user-registered conduit.
<u>CmGetCreatorTitle()</u>	Retrieves the display name of a user-registered conduit.
<u>CmGetCreatorType()</u>	(<i>Deprecated</i>) Returns the type—component or application—of the specified conduit.
<u>CmGetCreatorUser()</u>	(<i>Deprecated</i>) Retrieves the value of the Username configuration entry for the specified conduit.
<u>CmGetCreatorValueDword()</u>	Retrieves a DWORD configuration entry value for a user-registered conduit.
<u>CmGetCreatorValueString()</u>	Retrieves a string configuration entry value for a user-registered conduit.
<u>CmGetDiscoveryInfoByIndex()</u>	Retrieves information about how HotSync Manager discovers a user-registered conduit.
<u>CmGetHotSyncExecPath()</u>	(<i>Deprecated</i>) Retrieves the path and filename of the HotSync Manager executable on the desktop computer. (Strip off the HotSync .exe filename to get the path of support DLLs.)
<u>CmGetLibVersion()</u>	Returns the version number of the Conduit Manager API from the DLL that is currently loaded.

Conduit Manager API

Conduit Manager Functions

[CmGetNotifierDll\(\)](#)

(*Deprecated*) Retrieves the name of a notifier DLL.

[CmGetPCIdentifier\(\)](#)

Retrieves the unique identifier for the current Windows user on the desktop computer.

[CmGetSystemBackupConduit\(\)](#)

Retrieves the name of the conduit registered as the backup conduit for the system.

[CmGetSystemConduitByCreator\(\)](#)

Retrieves information about a conduit registered for the system, given the conduit's creator ID.

[CmGetSystemConduitByIndex\(\)](#)

Retrieves information about a conduit registered for the system given the conduit's index.

[CmGetSystemConduitCount\(\)](#)

Returns the number of conduits that are registered with HotSync Manager for the system.

[CmGetSystemConduitCreatorID\(\)](#)

Retrieves the creator ID of a conduit that is registered for the system.

[CmGetSystemCreatorDirectory\(\)](#)

Retrieves the name of the directory in which a system-registered conduit stores its user data.

[CmGetSystemCreatorFile\(\)](#)

Retrieves the name of the data file for a system-registered conduit.

[CmGetSystemCreatorIDLList\(\)](#)

Retrieves a list of the creator IDs of all conduits registered for the system.

<u>CmGetSystemCreatorName()</u>	Retrieves the filename of a system-registered conduit.
<u>CmGetSystemCreatorPriority()</u>	Retrieves the execution priority for a system-registered conduit.
<u>CmGetSystemCreatorRemote()</u>	Retrieves the name of a database on the handheld that is associated with a system-registered conduit.
<u>CmGetSystemCreatorTitle()</u>	Retrieves the display name of a system-registered conduit.
<u>CmGetSystemCreatorValueDword()</u>	Retrieves a DWORD configuration entry value for a system-registered conduit.
<u>CmGetSystemCreatorValueString()</u>	Retrieves a string configuration entry value for a system-registered conduit.
<u>CmGetSystemDiscoveryInfoByIndex()</u>	Retrieves information about how HotSync Manager discovers a system-registered conduit.
<u>CmGetSystemHotSyncExecPath()</u>	Retrieves the path and filename of the HotSync Manager executable on the desktop computer. (Strip off the HotSync .exe filename to get the path of support DLLs.)
<u>CmInstallConduit()</u>	Registers a conduit with HotSync Manager for the current Windows user. Uses a CmConduitType structure.

Conduit Manager API

Conduit Manager Functions

[CmInstallConduitByStruct\(\)](#)

Registers a conduit with HotSync Manager for the current Windows user. Uses a CmConduitType2 structure.

[CmInstallCreator\(\)](#)

Registers a new conduit creator ID with HotSync Manager for the current Windows user (requires calling other CmSetCreator...() functions to complete registration).

[CmInstallSystemConduitByStruct\(\)](#)

Registers a conduit with HotSync Manager for the system. Uses a CmConduitType2 structure.

[CmInstallSystemCreator\(\)](#)

Registers a new conduit creator ID with HotSync Manager for system (requires calling other CmSetSystemCreator...() functions to complete registration).

[CmIsCurrentUserAdmin\(\)](#)

Determines whether the current user has Windows administrator privileges.

[CmRemoveConduitByCreatorID\(\)](#)

Unregisters a conduit that is registered for the current Windows user, given the conduit's creator ID.

[CmRemoveConduitByIndex\(\)](#)

Unregisters a conduit that is registered for the current Windows user, given the conduit's index.

[CmRemoveSystemConduitByCreatorID\(\)](#)

Unregisters a conduit that is registered for the system, given the conduit's creator ID.

[CmRemoveSystemConduitByIndex\(\)](#)

Unregisters a conduit that is registered for the system, given the conduit's index.

[CmRestoreHotSyncSettings\(\)](#)

(*Deprecated*) Restores HotSync Manager configuration entry settings to their default values.

[CmSetBackupConduit\(\)](#)

Sets the path or filename of the backup conduit for the current Windows user.

[CmSetComPort\(\)](#)

Sets the name of the COM port that HotSync Manager uses for the specified type of connection on the desktop computer.

[CmSetCorePath\(\)](#)

(*Deprecated*) Sets the path of the HotSync users directory for the current Windows user.

[CmSetCreatorArgument\(\)](#)

(*Deprecated*) Sets the value of the Arguments configuration entry for the specified conduit.

[CmSetCreatorDirectory\(\)](#)

Sets the name of the directory in which a user-registered conduit stores its user data.

[CmSetCreatorFile\(\)](#)

Sets the name of the data file for a user-registered conduit.

[CmSetCreatorInfo\(\)](#)

(*Deprecated*) Sets a string associated with a user-registered conduit.

[CmSetCreatorIntegrate\(\)](#)

(*Deprecated*) Sets the value of the Integrate configuration entry for the specified conduit.

Conduit Manager API

Conduit Manager Functions

<u>CmSetCreatorModule()</u>	(<i>Deprecated</i>) Sets the value of the Module configuration entry for the specified conduit.
<u>CmSetCreatorName()</u>	Sets the filename of a user-registered conduit.
<u>CmSetCreatorPriority()</u>	Sets the execution priority for a user-registered conduit.
<u>CmSetCreatorRemote()</u>	Sets the name of a database on the handheld that is associated with a user-registered conduit.
<u>CmSetCreatorTitle()</u>	Sets the display name of a user-registered conduit.
<u>CmSetCreatorUser()</u>	(<i>Deprecated</i>) Sets the value of the Username configuration entry for the specified conduit.
<u>CmSetCreatorValueDword()</u>	Sets the value of a DWORD configuration entry for a user-registered conduit.
<u>CmSetCreatorValueString()</u>	Sets the value of a string configuration entry for a user-registered conduit
<u>CmSetHotSyncExecPath()</u>	(<i>Deprecated</i>) Sets the path and filename of the HotSync Manager executable for the current Windows user.
<u>CmSetNotifierDll()</u>	(<i>Deprecated</i>) Sets the name of a notifier DLL.
<u>CmSetPCIdentifier()</u>	(<i>Deprecated</i>) Sets the unique identifier for the current Windows user on the desktop computer.
<u>CmSetSystemBackupConduit()</u>	Sets the path or filename of the backup conduit for the system.

<u>CmSetSystemCreatorDirectory()</u>	Sets the name of the directory in which a system-registered conduit stores its user data.
<u>CmSetSystemCreatorFile()</u>	Sets the name of the data file for a system-registered conduit.
<u>CmSetSystemCreatorName()</u>	Sets the filename of a system-registered conduit.
<u>CmSetSystemCreatorPriority()</u>	Sets the execution priority for a system-registered conduit.
<u>CmSetSystemCreatorRemote()</u>	Sets the name of a database on the handheld that is associated with a system-registered conduit.
<u>CmSetSystemCreatorTitle()</u>	Sets the display name of a system-registered conduit.
<u>CmSetSystemCreatorValueDword()</u>	Sets the value of a DWORD configuration entry for a system-registered conduit.
<u>CmSetSystemCreatorValueString()</u>	Sets the value of a string configuration entry for a system-registered conduit
<u>FmDisableCurrentUserConduitByIndex()</u>	Disables a conduit that is in the current Windows user's Conduits folder, given the conduit's index.
<u>FmDisableCurrentUserConduitByPath()</u>	Disables a conduit that is in the current Windows user's Conduits folder, given the conduit's path or filename.
<u>FmDisableSystemConduitByIndex()</u>	Disables a conduit that is in the system's Conduits folder, given the conduit's index.

Conduit Manager API

Conduit Manager Functions

[FmDisableSystemConduitByPath\(\)](#)

Disables a conduit that is in the system's Conduits folder, given the conduit's path or filename.

[FmEnableCurrentUserConduitByPath\(\)](#)

Enables a conduit that is in the current Windows user's Disabled folder, given the conduit's path or filename.

[FmEnableSystemConduitByPath\(\)](#)

Enables a conduit that is in the system's Disabled folder, given the conduit's path or filename.

[FmGetCurrentUserConduitByIndex\(\)](#)

Retrieves information about a folder-based conduit registered for the current Windows user, given the conduit's index.

[FmGetCurrentUserConduitCount\(\)](#)

Returns the number of folder-based conduits that are registered with HotSync Manager for the current Windows user.

[FmGetCurrentUserConduitFolder\(\)](#)

Retrieves the path of the current Windows user's Conduits folder.

[FmGetCurrentUserDisabledConduitFolder\(\)](#)

Retrieves the path of the current Windows user's Disabled folder.

[FmGetSystemConduitByIndex\(\)](#)

Retrieves information about a folder-based conduit registered for the system, given the conduit's index.

[FmGetSystemConduitCount\(\)](#)

Returns the number of folder-based conduits that are registered with HotSync Manager for the system.

[FmGetSystemConduitFolder\(\)](#)

Retrieves the path of the system's Conduits folder.

[FmGetSystemDisabledConduitFolder\(\)](#)

Retrieves the path of the system's Disabled folder.

Conduit Manager API

CmConvertCreatorIDToString

CmConvertCreatorIDToString Function

Purpose Converts a conduit creator ID from a DWORD to a string.

Declared In CondMgr.h

Prototype int CmConvertCreatorIDToString (DWORD *dwID*,
 TCHAR **pString*, int **piSize*)

Parameters → *dwID*
 A creator ID as a DWORD.

← *pString*
 A pointer to a character buffer. Upon return, this contains the creator ID as a string.

↔ *piSize*
 A pointer to an integer value that specifies the size of the character buffer referenced by *pString*. Upon return, this is the actual size, in TCHARs of the creator ID string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_BUFFER_TOO_SMALL

The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the *piSize* parameter.

ERR_INVALID_CREATOR_ID

The specified creator ID is not a valid creator ID.

ERR_INVALID_POINTER

The specified pointer is not a valid pointer.

For descriptions of all Conduit Manager error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [CmConvertStringToCreatorID\(\)](#)

CmConvertStringToCreatorID Function

Purpose	Converts a conduit creator ID from a string to a DWORD.
Declared In	CondMgr.h
Prototype	<pre>int CmConvertStringToCreatorID (const TCHAR *pString, DWORD *pdwID)</pre>
Parameters	<p>→ <i>pString</i> A creator ID as a string. This must be a four-character creator ID.</p> <p>← <i>pdwID</i> The creator ID converted to a DWORD.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_INVALID_CREATOR_ID The specified creator ID is not a valid creator ID.</p> <p>ERR_INVALID_POINTER The specified pointer is not a valid pointer.</p> <p>For descriptions of all Conduit Manager error codes, see “Conduit Manager Error Codes” on page 843.</p>
Comments	Other Conduit Manager functions take a creator ID as a DWORD value rather than as a string.
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	CmConvertCreatorIDToString()

Conduit Manager API

CmGetBackupConduit

CmGetBackupConduit Function

Purpose	Retrieves the path or filename of the backup conduit for the current Windows user.
Declared In	CondMgr.h
Prototype	<pre>int CmGetBackupConduit (TCHAR *pConduit, int *piSize)</pre>
Parameters	<p>← <i>pConduit</i> A pointer to a character buffer. Upon return, this contains the full path or filename of the conduit used as the backup conduit.</p> <p>↔ <i>piSize</i> A pointer to an integer value that specifies the size of the character buffer referenced by <i>pConduit</i>. Upon return, this is the actual size, in TCHARS, of the backup conduit name string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_REGISTRY_ACCESS Unable to access the conduit configuration entries.</p> <p>ERR_BUFFER_TOO_SMALL The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the <i>piSize</i> parameter.</p> <p>ERR_INVALID_POINTER The specified pointer is not a valid pointer.</p>
	For descriptions of all Conduit Manager error codes, see “ Conduit Manager Error Codes ” on page 843.

Comments	This function retrieves the value of the HotSync Manager\BackupConduit configuration entry used by HotSync Manager for the current Windows user. To retrieve this information for the backup conduit that is registered for the system, call CmGetSystemBackupConduit() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	HotSync Manager\BackupConduit , CmSetBackupConduit() , CmGetSystemBackupConduit()

Conduit Manager API

CmGetComPort

CmGetComPort Function

Purpose	Retrieves the name of the COM port that HotSync Manager uses for the specified type of connection on the desktop computer.
Declared In	CondMgr.h
Prototype	<pre>int CmGetComPort (int iType, TCHAR *pPort, int *piSize)</pre>
Parameters	<p>→ <i>iType</i> The type of connection for which to retrieve the COM port. You must specify one of the values described in “Communications Ports” on page 662.</p> <p>← <i>pPort</i> A character buffer. Upon return, this contains the port name—for example, “COM1”.</p> <p>↔ <i>piSize</i> A pointer to an integer that specifies the size, in TCHARs of the buffer referenced by the <i>pPort</i> parameter. Upon return, this is the actual size, in TCHARs of the port name string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_REGISTRY_ACCESS Unable to access the conduit configuration entries.</p> <p>ERR_BUFFER_TOO_SMALL The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the <i>piSize</i> parameter.</p> <p>ERR_INVALID_POINTER The specified pointer is not a valid pointer.</p> <p>ERR_INVALID_COM_PORT_TYPE The specified communications port type is not valid. You must use one of the values described in “Communications Ports” on page 662.</p>

For descriptions of all Conduit Manager error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments	This function retrieves either the HotSync Manager\ModemComPort or HotSync Manager\DirectComPort entry from the HotSync Manager portion of the configuration entries, depending on the value you specify for the <i>iType</i> parameter. Note that these values are set for the current Windows user only. There are no corresponding settings for the system. HotSync Manager sets default values for each new Windows users.
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	HotSync Manager\ModemComPort , HotSync Manager\DirectComPort , CmSetComPort()

Conduit Manager API

CmGetConduitByCreator

CmGetConduitByCreator Function

Purpose	Retrieves information about a conduit registered for the current Windows user, given the conduit's creator ID.
Declared In	CondMgr.h
Prototype	<pre>int CmGetConduitByCreator (const char *pCreatorID, HANDLE *hStruct)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID of a conduit, provided as a string.</p> <p>← <i>hStruct</i> A pointer to a handle. This function allocates storage for a CmConduitType structure, stores data into the structure, and returns the handle. It is your responsibility to deallocate the storage.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_NO_MEMORY Not enough memory is available to perform the requested operation.</p> <p>ERR_REGISTRY_ACCESS Unable to access the conduit configuration entries.</p> <p>ERR_INVALID_CREATOR_ID The specified creator ID is not a valid creator ID.</p> <p>ERR_INVALID_POINTER The specified pointer is not a valid pointer.</p> <p>For descriptions of all Conduit Manager error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments Because this function allocates storage and returns a handle for that storage, you must deallocate the handle when you have finished.

Note that Conduit Manager allocates the *hStruct* handle by calling the `GlobalAlloc()` function with the `GMEM_MOVEABLE` flag. To convert the handle into a pointer, you need to use the `GlobalLock()` and `GlobalUnlock()` functions. For example:

```
CmConduitType *pConduit =  
    (CmConduitType*) GlobalLock(*hStruct);  
GlobalUnlock(*hStruct);
```

This function retrieves information about a conduit that is registered for the current Windows user. To retrieve this information for a conduit that is registered for the system, call [CmGetSystemConduitByCreator\(\)](#).

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [CmConduitType](#), [CmInstallConduitByStruct\(\)](#),
[CmGetConduitByIndex\(\)](#),
[CmGetSystemConduitByCreator\(\)](#)

Conduit Manager API

CmGetConduitByIndex

CmGetConduitByIndex Function

Purpose	Retrieves information about a conduit registered for the current Windows user, given the conduit's index.
Declared In	CondMgr.h
Prototype	<pre>int CmGetConduitByIndex (int iIndex, CmConduitType2 &sConduitInfo)</pre>
Parameters	<p>→ <i>iIndex</i> The zero-based index of a conduit registered for the current Windows user. Call CmGetConduitCount() to determine the highest index value.</p> <p>← <i>sConduitInfo</i> A reference to a CmConduitType2 structure that describes the specified conduit.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_INDEX_OUT_OF_RANGE The specified index value is out of range.</p> <p>For descriptions of all Conduit Manager error codes, see “Conduit Manager Error Codes” on page 843.</p>
Comments	This function retrieves information about a conduit that is registered for the current Windows user. To retrieve this information for a conduit that is registered for the system, call CmGetSystemConduitByIndex() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	CmConduitType2 , CmInstallConduitByStruct() , CmGetConduitCount() , CmGetSystemConduitByIndex() , CmGetConduitByCreator()

CmGetConduitCount Function

Purpose	Returns the number of conduits that are registered with HotSync Manager for the current Windows user.
Declared In	CondMgr.h
Prototype	<code>int CmGetConduitCount (void)</code>
Parameters	None.
Returns	If successful, returns the number of registered conduits. If unsuccessful, returns the following nonzero error code value: <code>ERR_REGISTRY_ACCESS</code> For descriptions of all Conduit Manager error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	The returned count includes all conduits that are registered for the current Windows user. This count includes conduits registered by placement in the user’s Conduits folder (folder-registered) and those registered by configuration entries (conventionally registered). This count does not include backup or install conduits, nor does it include system-registered conduits. To retrieve this information for conduits that are registered for the system, call CmGetSystemConduitCount () . You can use this count to determine the range of valid index values that you can use with CmGetConduitByIndex () .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	CmGetSystemConduitCount () , CmGetConduitByIndex ()

Conduit Manager API

CmGetConduitCreatorID

CmGetConduitCreatorID Function

Purpose	Retrieves the creator ID of a conduit that is registered for the current Windows user.
Declared In	CondMgr.h
Prototype	<pre>int CmGetConduitCreatorID (int iIndex, char *pCreatorID, int *piSize)</pre>
Parameters	<p>→ <i>iIndex</i> The zero-based index of a conduit registered for the current Windows user. Call CmGetConduitCount() to determine the highest index value.</p> <p>← <i>pCreatorID</i> A character buffer. Upon return, this contains the creator ID string.</p> <p>↔ <i>piSize</i> A pointer to an integer that specifies the size, in chars, of the buffer referenced by the <i>pCreatorID</i> parameter. Upon return, this is the actual size, in chars, of the creator ID string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_REGISTRY_ACCESS ERR_BUFFER_TOO_SMALL The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the <i>piSize</i> parameter.</p> <p>For descriptions of all Conduit Manager error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments	Use 0 to specify the first conduit defined in the configuration entries, and increment the index value by 1 for each subsequent conduit. This function retrieves information about a conduit that is registered for the current Windows user. To retrieve this information for a conduit that is registered for the system, call <u>CmGetSystemConduitCreatorID()</u> .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	<u>CmGetConduitCount()</u> , <u>CmGetSystemConduitCreatorID()</u> , <u>CmInstallConduitByStruct()</u>

Conduit Manager API

CmGetCorePath

CmGetCorePath Function

Purpose	Retrieves the path of the HotSync users' directory for the current Windows user.
Declared In	CondMgr.h
Prototype	<code>int CmGetCorePath (TCHAR *<i>pPath</i>, int *<i>piSize</i>)</code>
Parameters	<p>$\leftarrow pPath$ A pointer to a character buffer. Upon return, it contains the path.</p> <p>$\leftrightarrow piSize$ A pointer to an integer that specifies the size, in TCHARS of the buffer referenced by the <i>pPath</i> parameter. Upon return, this is the actual size, in TCHARS, of the path string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p><code>ERR_BUFFER_TOO_SMALL</code> The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the <i>piSize</i> parameter.</p> <p><code>ERR_REGISTRY_ACCESS</code></p> <p><code>ERR_INVALID_POINTER</code></p> <p>For descriptions of all Conduit Manager error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments This function retrieves the value of the [Core\Path](#) configuration entry for the current Windows user. Because this path may differ for each Windows user, there is no corresponding “system-level” path. Conduit Manager versions 3 and later are aware of multiple Windows users, so each Windows user must have a separate Core\Path value. To meet Windows standards for the placement of user data, a typical value is C:\Documents and Settings\<WinUsername>\My Documents\Palm OS Desktop. If you need the full path of a HotSync user’s directory, then call [Um GetUserDirectory\(\)](#) and append that path to the Core\Path value.

NOTE: The Core Path returned by `CmGetCorePath()` is *not* necessarily the location of CondMgr.dll, Instaide.dll, or any of the other support DLLs. These DLLs can be in a directory different from that of the user directories. Use `CmGetCorePath()` to determine the location of HotSync user directories only.

To determine the path of HotSync Manager, CondMgr.dll, Instaide.dll, and other DLLs, call [CmGetHotSyncExecPath\(\)](#) instead.

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [CmGetHotSyncExecPath\(\)](#)

Conduit Manager API

CmGetCreatorArgument

CmGetCreatorArgument Function

Purpose	(<i>Degraded</i>) Retrieves the value of the Arguments configuration entry for the specified conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmGetCreatorArgument (const char *pCreatorID, TCHAR *pArgument, int *piSize)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Arguments value you want to retrieve.</p> <p>← <i>pArgument</i> A pointer to a character buffer. Upon return, this is the value of the Arguments configuration entry for the specified conduit.</p> <p>↔ <i>piSize</i> A pointer to an integer that specifies the size, in TCHARs, of the buffer referenced by the <i>pArgument</i> parameter. Upon return, this is the actual size, in TCHARs, of the Arguments string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_BUFFER_TOO_SMALL The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the <i>piSize</i> parameter.</p> <p>ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments	As of CDK 6.0, this function is deprecated. No version of HotSync Manager uses this value.
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	Arguments , CmSetCreatorArgument ()

Conduit Manager API

CmGetCreatorDirectory

CmGetCreatorDirectory Function

Purpose	Retrieves the name of the directory in which a user-registered conduit stores its user data.
Declared In	CondMgr.h
Prototype	<pre>int CmGetCreatorDirectory (const char *pCreatorID, TCHAR *pDirectory, int *piSize)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Directory value you want to retrieve.</p> <p>← <i>pDirectory</i> A pointer to a character buffer. Upon return, this is the value of the Directory configuration entry for the specified conduit.</p> <p>↔ <i>piSize</i> A pointer to an integer that specifies the size, in TCHARs, of the buffer referenced by the <i>pDirectory</i> parameter. Upon return, this is the actual size, in TCHARs, of the Directory string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_BUFFER_TOO_SMALL The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the <i>piSize</i> parameter.</p> <p>ERR_VALUE_NOT_FOUND ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER</p>

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments

This function retrieves the value of the [Directory](#) configuration entry for the specified conduit. The directory string specifies the directory, under the current HotSync user’s directory, that the conduit creates to store its user data and support files. For example, if the value of Directory were DateBook and the value of [Core\Path](#) were a typical value, then the full path to this conduit’s data directory is:

```
C:\Documents and Settings\<WinUsername>\  
My Documents\Palm OS Desktop\  
<HotSyncUsername>\DateBook
```

This function retrieves information about a conduit that is registered for the current Windows user. To retrieve this information for a conduit that is registered for the system, call [CmGetSystemCreatorDirectory\(\)](#).

Compatibility

Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also

[Directory](#), [CmSetCreatorDirectory\(\)](#),
[CmGetSystemCreatorDirectory\(\)](#)

Conduit Manager API

CmGetCreatorFile

CmGetCreatorFile Function

Purpose Retrieves the name of the data file for a user-registered conduit.

Declared In CondMgr.h

Prototype int CmGetCreatorFile (const char **pCreatorID*,
 TCHAR **pFile*, int **piSize*)

Parameters → *pCreatorID*
The creator ID string of the conduit whose [File](#) value you want to retrieve.

← *pFile*
A pointer to a character buffer. Upon return, this is the value of the [File](#) configuration entry for the specified conduit.

↔ *piSize*
A pointer to an integer that specifies the size, in TCHARs, of the buffer referenced by the *pFile* parameter. Upon return, this is the actual size, in TCHARs, of the [File](#) string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_AMBIGUOUS_CREATORID

ERR_NO_CONDUIT

ERR_REGISTRY_ACCESS

ERR_BUFFER_TOO_SMALL

The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the *piSize* parameter.

ERR_VALUE_NOT_FOUND

ERR_INVALID_CREATOR_ID

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments This function retrieves the value of the [File](#) configuration entry for the specified conduit. The file string specifies the name of the desktop file to synchronize with the handheld database. This file can be anywhere, but it is commonly located in the directory returned by [CmGetCreatorDirectory\(\)](#).

This function retrieves information about a conduit that is registered for the current Windows user. To retrieve this information for a conduit that is registered for the system, call [CmGetSystemCreatorFile\(\)](#).

Comments Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [File](#), [CmSetCreatorFile\(\)](#), [CmGetSystemCreatorFile\(\)](#)

Conduit Manager API

CmGetCreatorIDList

CmGetCreatorIDList Function

Purpose	Retrieves a list of the creator IDs of all conduits registered for the current Windows user.
Declared In	CondMgr.h
Prototype	<pre>int CmGetCreatorIDList (CM_CREATORLIST_TYPE *pCreatorList, int *piSize)</pre>
Parameters	<p>→ <i>pCreatorList</i> A pointer to an array of CM_CREATORLIST_ITEM_TYPE structures containing creator IDs.</p> <p>↔ <i>piSize</i> A pointer to an integer value that specifies the number of entries in the array referenced by the <i>pCreatorList</i> parameter. Upon return, this is the actual number of array items that were filled in. If this function fails because the array is too small, the value of <i>piSize</i> upon return is the required number of array items.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_REGISTRY_ACCESS ERR_BUFFER_TOO_SMALL The size of the array that you supplied is too small to contain the number of creator ID items, so this function passes back the required number of items into the <i>piSize</i> parameter.</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments	This function returns the list of creator IDs via a CM_CREATORLIST_TYPE pointer to an array of structures of type CM_CREATORLIST_ITEM_TYPE . This function retrieves information about conduits that are registered for the current Windows user. To retrieve this information for conduits that are registered for the system, call CmGetSystemCreatorIDList() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	CM_CREATORLIST_ITEM_TYPE , CmGetSystemCreatorIDList() , CmInstallConduitByStruct()

Conduit Manager API

CmGetCreatorInfo

CmGetCreatorInfo Function

Purpose	(<i>Degraded</i>) Retrieves a string associated with a user-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmGetCreatorInfo (const char *pCreatorID, TCHAR *pInfo, int *piSize)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Information value you want to retrieve.</p> <p>← <i>pInfo</i> A pointer to a character buffer. Upon return, this is the value of the Information configuration entry for the specified conduit.</p> <p>↔ <i>piSize</i> A pointer to an integer that specifies the size, in TCHARs, of the buffer referenced by the <i>pInfo</i> parameter. Upon return, this is the actual size, in TCHARs, of the Information string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_BUFFER_TOO_SMALL The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the <i>piSize</i> parameter. ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.

Comments	As of CDK 6.0, this function is deprecated. No version of HotSync Manager uses this value.
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	Information , CmSetCreatorInfo()

Conduit Manager API

CmGetCreatorIntegrate

CmGetCreatorIntegrate Function

Purpose	(<i>Degraded</i>) Retrieves the value of the Integrate configuration entry for the specified conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmGetCreatorIntegrate (const char *pCreatorID, DWORD *pdwIntegrate)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Integrate value you want to retrieve.</p> <p>← <i>pdwIntegrate</i> A pointer to a DWORD value. Upon return, this is the value of the Integrate configuration entry for the specified conduit.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	As of CDK 6.0, this function is deprecated. No version of HotSync Manager uses this value.
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	Integrate , CmSetCreatorIntegrate()

CmGetCreatorModule Function

Purpose (*Deprecated*) Retrieves the value of the [Module](#) configuration entry for the specified conduit.

Declared In CondMgr.h

Prototype

```
int CmGetCreatorModule (const char *pCreatorID,  
                      TCHAR *pModule, int *piSize)
```

Parameters

→ *pCreatorID*

The creator ID string of the conduit whose [Module](#) value you want to retrieve.

← *pModule*

A pointer to a character buffer. Upon return, this is the value of the [Module](#) configuration entry for the specified conduit.

↔ *piSize*

The size, in TCHARS, of the buffer referenced by the *pModule* parameter. Upon return, this is the actual size, in TCHARS, of the [Module](#) string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_AMBIGUOUS_CREATORID

ERR_NO_CONDUIT

ERR_REGISTRY_ACCESS

ERR_BUFFER_TOO_SMALL

The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the *piSize* parameter.

ERR_INVALID_CREATOR_ID

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Conduit Manager API

CmGetCreatorModule

Comments	As of CDK 6.0, this function is deprecated. No version of HotSync Manager uses this value.
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	Module , CmSetCreatorModule()

CmGetCreatorName Function

Purpose Retrieves the filename of a user-registered conduit.

Declared In CondMgr.h

Prototype int CmGetCreatorName (const char **pCreatorID*,
 TCHAR **pConduitName*, int **piSize*)

Parameters → *pCreatorID*
The creator ID string of the conduit whose [Conduit](#) value you want to retrieve.

← *pConduitName*
A pointer to a character buffer. Upon return, this is the value of the [Conduit](#) configuration entry for the specified conduit.

↔ *piSize*
A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the *pConduitName* parameter.
Upon return, this is the actual size, in TCHARS, of the [Conduit](#) string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_AMBIGUOUS_CREATORID

ERR_NO_CONDUIT

ERR_REGISTRY_ACCESS

ERR_BUFFER_TOO_SMALL

The buffer that you supplied is too small to contain the string, so this function stores the actual required size into the *piSize* parameter.

ERR_INVALID_CREATOR_ID

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Conduit Manager API

CmGetCreatorName

Comments	This function retrieves the value of the Conduit configuration entry for the specified conduit. This value is the filename of the conduit DLL that is registered with the specified creator ID—for example, conduit filename ToDoCond.dll is registered with the creator ID 'todo'. This function retrieves information about a conduit that is registered for the current Windows user. To retrieve this information for a conduit that is registered for the system, call CmGetSystemCreatorName () .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	Conduit , CmSetCreatorName () , CmGetSystemCreatorName ()

CmGetCreatorPriority Function

Purpose	Retrieves the execution priority for a user-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmGetCreatorPriority (const char *pCreatorID, DWORD *pdwPriority)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Priority value you want to retrieve.</p> <p>← <i>pdwPriority</i> A pointer to a DWORD value. Upon return, this is the value of the Priority configuration entry for the specified conduit, which is a value between 0 and 4.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER ERR_VALUE_NOT_FOUND</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Conduit Manager API

CmGetCreatorPriority

- Comments** This function retrieves the value of the [Priority](#) configuration entry for the specified conduit. HotSync Manager uses the priority value to determine the order in which it executes conduits—that is, the order in which it calls conduits' `OpenConduit()` entry points. HotSync Manager runs conduits with a value of 0 first and those with 4 last.
- This function retrieves information about a conduit that is registered for the current Windows user. To retrieve this information for a conduit that is registered for the system, call [CmGetSystemCreatorPriority\(\)](#).
- Compatibility** Conduit Manager versions: 2 and later.
Palm OS versions: All.
- See Also** [Priority](#), [CmSetCreatorPriority\(\)](#),
[CmGetSystemCreatorPriority\(\)](#)

CmGetCreatorRemote Function

Purpose Retrieves the name of a database on the handheld that is associated with a user-registered conduit.

Declared In CondMgr.h

Prototype

```
int CmGetCreatorRemote (const char *pCreatorID,  
                      TCHAR *pRemoteDB, int *piSize)
```

Parameters

→ *pCreatorID*

The creator ID string of the conduit whose [Remote](#) value you want to retrieve.

← *pRemoteDB*

A pointer to a character buffer. Upon return, this is the value of the [Remote](#) configuration entry for the specified conduit.

↔ *piSize*

A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the *pRemoteDB* parameter. Upon return, this is the actual size, in TCHARS, of the [Remote](#) string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_AMBIGUOUS_CREATORID

ERR_NO_CONDUIT

ERR_REGISTRY_ACCESS

ERR_BUFFER_TOO_SMALL

The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size into the *piSize* parameter.

ERR_VALUE_NOT_FOUND

ERR_INVALID_CREATOR_ID

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Conduit Manager API

CmGetCreatorRemote

Comments This function retrieves the value of the [Remote](#) configuration entry for the specified conduit. This value names a single database on the handheld that is associated with this conduit.

NOTE: Though a conduit can access multiple databases on the handheld, `CmGetCreatorRemote()` returns only one handheld database name: the one specified in the `Remote` configuration entry for this conduit. See the description of “[Remote](#)” on page 183 for more information to help you decide whether you want to use this configuration entry.

This function retrieves information about a conduit that is registered for the current Windows user. To retrieve this information for a conduit that is registered for the system, call [CmGetSystemCreatorRemote\(\)](#).

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [Remote](#), [CmSetCreatorRemote\(\)](#),
[CmGetSystemCreatorRemote\(\)](#)

CmGetCreatorTitle Function

Purpose Retrieves the display name of a user-registered conduit.

Declared In CondMgr.h

Prototype `int CmGetCreatorTitle (const char *pCreatorID,
 TCHAR *pTitle, int *piSize)`

Parameters → *pCreatorID*

The creator ID string of the conduit whose [Name](#) value you want to retrieve.

← *pTitle*

A pointer to a character buffer. Upon return, this is the value of the [Name](#) configuration entry for the specified conduit.

↔ *piSize*

A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the *pTitle* parameter. Upon return, this is the actual size, in TCHARS, of the [Name](#) string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

`ERR_AMBIGUOUS_CREATORID`

`ERR_NO_CONDUIT`

`ERR_REGISTRY_ACCESS`

`ERR_BUFFER_TOO_SMALL`

The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size into the *piSize* parameter.

`ERR_VALUE_NOT_FOUND`

`ERR_INVALID_CREATOR_ID`

`ERR_INVALID_POINTER`

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Conduit Manager API

CmGetCreatorTitle

Comments	This function retrieves the value of the Name configuration entry for the specified conduit. When this value is not NULL, HotSync Manager uses it as the conduit's display name in its Custom and HotSync Progress dialog boxes. If this value is NULL, HotSync Manager uses the value returned by a conduit's GetConduitInfo() entry point, and if that fails, its GetConduitName() entry point.
	This function retrieves information about a conduit that is registered for the current Windows user. To retrieve this information for a conduit that is registered for the system, call CmGetSystemCreatorTitle() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.

See Also [Name](#), [CmSetCreatorTitle\(\)](#), [CmGetSystemCreatorTitle\(\)](#)

CmGetCreatorType Function

Purpose (*Degraded*) Returns the type—component or application—of the specified conduit.

Declared In CondMgr.h

Prototype int CmGetCreatorType (const char **pCreator*)

Parameters → *pCreator*
The creator ID string of the conduit whose type value you want to retrieve.

Returns If successful, returns one of the conduit type constant values, as described in “[Conduit Information Types](#)” on page 661.

If unsuccessful, returns one of the following nonzero error code values:

ERR_NO_CONDUIT
ERR_REGISTRY_ACCESS
ERR_INVALID_CREATOR_ID
ERR_INVALID_POINTER

Comments As of CDK 6.0, this function is deprecated. No version of HotSync Manager uses this value.

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

Conduit Manager API

CmGetCreatorUser

CmGetCreatorUser Function

Purpose	(<i>Degraded</i>) Retrieves the value of the Username configuration entry for the specified conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmGetCreatorUser (const char *pCreatorID, TCHAR *pUsername, int *piSize)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Username value you want to retrieve.</p> <p>← <i>pUsername</i> A pointer to a character buffer. Upon return, this is the value of the Username configuration entry for the specified conduit.</p> <p>↔ <i>piSize</i> A pointer to an integer that specifies the size, in TCHARs, of the buffer referenced by the <i>pUsername</i> parameter. Upon return, this is the actual size, in TCHARs, of the Username string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_BUFFER_TOO_SMALL The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the <i>piSize</i> parameter.</p> <p>ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments	As of CDK 6.0, this function is deprecated. No version of HotSync Manager uses this value.
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	Username , CmSetCreatorUser()

Conduit Manager API

CmGetCreatorValueDword

CmGetCreatorValueDword Function

Purpose	Retrieves a DWORD configuration entry value for a user-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmGetCreatorValueDword (const char *pCreatorID, TCHAR *pValue, DWORD *dwValue, DWORD dwDefault)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit for which you want to retrieve a configuration entry value.</p> <p>→ <i>pValue</i> The configuration entry name.</p> <p>← <i>dwValue</i> Upon return, this contains the value of the configuration entry specified by <i>pValue</i>.</p> <p>→ <i>dwDefault</i> The default value to return in <i>dwValue</i> if the specified <i>pValue</i> name could not be found in the configuration entries.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER ERR_VALUE_NOT_FOUND</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments	This function retrieves a DWORD value from the configuration entries for a conduit. This is a general purpose function for retrieving a conduit configuration entry value by name. You can use it to retrieve values of configuration entries that you create for your own use with CmSetCreatorValueDword() . If Conduit Manager does not find the configuration entry name specified by <i>pValue</i> , it sets <i>dwValue</i> equal to the value of <i>dwDefault</i> and returns 0 (success). However, if the conduit specified by <i>pCreatorID</i> does not exist, this function does <i>not</i> set <i>dwValue</i> equal to the value of <i>dwDefault</i> but does return an error. This function retrieves information about a conduit that is registered for the current Windows user. To retrieve this information for a conduit that is registered for the system, call CmGetSystemCreatorValueDword() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	CmSetCreatorValueDword() , CmGetSystemCreatorValueDword() , CmGetCreatorValueString()

Conduit Manager API

CmGetCreatorValueString

CmGetCreatorValueString Function

Purpose	Retrieves a string configuration entry value for a user-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmGetCreatorValueString (const char *pCreatorID, TCHAR *pValue, TCHAR *pString, int *piSize, TCHAR *pDefault)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit for which you want to retrieve a configuration entry value.</p> <p>→ <i>pValue</i> The configuration entry name.</p> <p>← <i>pString</i> A pointer to a character buffer. Upon return, this is the value of the configuration entry specified by <i>pValue</i>.</p> <p>↔ <i>piSize</i> A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the <i>pString</i> parameter. Upon return, this is the actual size, in TCHARS, of the string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p> <p>→ <i>pDefault</i> The default value to return in <i>pString</i> if the specified <i>pValue</i> name could not be found in the configuration entries.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_BUFFER_TOO_SMALL The size of the buffer that you supplied is too small to contain the string, so this function passes back the actual required size into the <i>piSize</i> parameter.</p>

ERR_INVALID_CREATOR_ID

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments

This function retrieves a string value from the configuration entries for a conduit. This is a general purpose function for retrieving a conduit configuration entry value by name. You can use it to retrieve values of configuration entries that you create for your own use with [CmSetCreatorValueString\(\)](#).

If Conduit Manager does not find the configuration entry name specified by *pValue*, it passes back the default value (sets *pString* equal to the value of *pDefault*) and returns 0 (success). However, if the conduit specified by *pCreatorID* does not exist, Conduit Manager does *not* pass back the default value but does return an error.

This function retrieves information about a conduit that is registered for the current Windows user. To retrieve this information for a conduit that is registered for the system, call [CmGetSystemCreatorValueString\(\)](#).

Compatibility

Conduit Manager versions: 2 and later.

Palm OS versions: All.

See Also

[CmSetCreatorValueString\(\)](#),
[CmGetSystemCreatorValueString\(\)](#),
[CmGetCreatorValueDword\(\)](#)

Conduit Manager API

CmGetDiscoveryInfoByIndex

CmGetDiscoveryInfoByIndex Function

Purpose Retrieves information about how HotSync Manager discovers a user-registered conduit.

Declared In CondMgr.h

Prototype int CmGetDiscoveryInfoByIndex (int *iIndex*,
CmDiscoveryInfoType &*sDiscoveryInfo*)

Parameters → *iIndex*
The zero-based index of a conduit registered for the current Windows user. Call [CmGetConduitCount\(\)](#) to determine the highest index value.

← *sDiscoveryInfo*
A reference to a [CmDiscoveryInfoType](#) structure that describes whether HotSync Manager can discover the specified conduit and if so, whether the conduit is in a Conduits folder or in the conduit configuration entries.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_INDEX_OUT_OF_RANGE

The specified index value is out of range.

Comments Note that if the bLoadable field of the [CmDiscoveryInfoType](#) structure is FALSE, then the specified conduit is not properly registered and HotSync Manager cannot call it.

This function retrieves information about a conduit that is registered for the current Windows user. To retrieve this information for a conduit that is registered for the system, call [CmGetSystemDiscoveryInfoByIndex\(\)](#).

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [CmDiscoveryInfoType](#),
[CmGetSystemDiscoveryInfoByIndex\(\)](#)

CmGetHotSyncExecPath Function

Purpose (*Deprecated*) Retrieves the path and filename of the HotSync Manager executable on the desktop computer. (Strip off the HotSync.exe filename to get the path of support DLLs.)

Declared In CondMgr.h

Prototype int CmGetHotSyncExecPath (TCHAR **pPath*,
 int **piSize*)

Parameters ← *pPath*
A pointer to a character buffer. Upon return, this is the path and filename of the HotSync Manager executable—for example, C:\Program Files\PalmSource\Desktop\HotSync.exe. If HotSync Manager is not installed, this parameter points to an empty string.

↔ *piSize*

A pointer to an integer that specifies the size, in TCHARs, of the buffer referenced by the *pPath* parameter. Upon return, this is the actual size, in TCHARs, of the string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_REGISTRY_ACCESS

ERR_BUFFER_TOO_SMALL

The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size into the *piSize* parameter.

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Conduit Manager API

CmGetHotSyncExecPath

Comments	As of CDK 6.0, this function is deprecated. However, it is still available for compatibility with legacy conduit installers. All others should use CmGetSystemHotSyncExecPath() instead, which is available in Conduit Manager versions 3 and later.
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	CmGetCorePath() , CmGetSystemHotSyncExecPath()

CmGetLibVersion Function

Purpose	Returns the version number of the Conduit Manager API from the DLL that is currently loaded.
Declared In	CondMgr.h
Prototype	WORD CmGetLibVersion ()
Parameters	None.
Returns	Returns the value of one of the CM_UPDATE... constants defined in " Conduit Manager Versions " on page 663.
Comments	The "Compatibility" sections in this chapter indicate in which versions of the Conduit Manager API that each function, structure, or constant is available. For example, "Conduit Manager versions: 2 and later" indicates that the API element is available if CmGetLibVersion () returns a value of 0x0002 or greater.
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.

Conduit Manager API

CmGetNotifierDll

CmGetNotifierDll Function

Purpose (*Deprecated*) Retrieves the name of a notifier DLL.

Declared In CondMgr.h

Prototype int CmGetNotifierDll (int *iIndex*,
 TCHAR **pNotifier*, int **piSize*)

Parameters

→ *iIndex*

The integer index of the notifier that you want to retrieve.

← *pNotifier*

A pointer to a character buffer. Upon return, this is the name of the notifier DLL.

↔ *piSize*

A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the *pNotifier* parameter. Upon return, this is the actual size, in TCHARS, of the string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_REGISTRY_ACCESS

ERR_BUFFER_TOO_SMALL

The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size into the *piSize* parameter.

ERR_VALUE_NOT_FOUND

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments This function retrieves the name of a notifier DLL that is stored as a value in the HotSync Manager portion of the configuration entries. The first notifier configuration entry is named Notifier0, the second entry is named Notifier1, and so on.

IMPORTANT: This function is deprecated. Use one of the Notifier Install Manager functions instead (see [Chapter 13, “Notifier Install Manager API,”](#) on page 909).

`CmGetNotifierDLL()` is retained for backward compatibility, but the Notifier Install Manager enables you more easily to install, uninstall, and modify notifiers.

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [NmGetByIndex\(\)](#) or other “[Notifier Install Manager API](#)” on page 909

Conduit Manager API

CmGetPCIdentifier

CmGetPCIdentifier Function

Purpose	Retrieves the unique identifier for the current Windows user on the desktop computer.
Declared In	CondMgr.h
Prototype	<code>int CmGetPCIdentifier (DWORD *pdwPCID)</code>
Parameters	$\rightarrow pdwPCID$ A pointer to a DWORD value. Upon return, this is the PC identification value.
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: <code>ERR_REGISTRY_ACCESS</code> <code>ERR_INVALID_POINTER</code> For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	This function retrieves the value of the HotSync Manager\PCIdentifier configuration entry for the current Windows user. HotSync Manager generates a different PC ID for each Windows user who runs HotSync Manager. Different HotSync users using the same Windows login have the same PC ID.
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	HotSync Manager\PCIdentifier , CmSetPCIdentifier()

CmGetSystemBackupConduit Function

Purpose Retrieves the name of the conduit registered as the backup conduit for the system.

Declared In CondMgr.h

Prototype

```
int CmGetSystemBackupConduit (TCHAR *pConduit,  
                             int *piSize)
```

Parameters

← *pConduit*
A pointer to a character buffer. Upon return, this contains the name of the conduit used as the system backup conduit on the desktop computer.

↔ *piSize*

A pointer to an integer value that specifies the size of the character buffer referenced by *pConduit*. Upon return, this is the actual size, in TCHARs, of the backup conduit name string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_REGISTRY_ACCESS

ERR_BUFFER_TOO_SMALL

The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the *piSize* parameter.

ERR_INVALID_POINTER

ERR_UNABLE_TO_SET_CONDUIT_VALUE

For descriptions of all Conduit Manager error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Conduit Manager API

CmGetSystemBackupConduit

Comments	This function retrieves the value of the HotSync Manager\BackupConduit configuration entry used by HotSync Manager for the system. To retrieve this information for the backup conduit that is registered for the current Windows user, call CmGetBackupConduit () .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	HotSync Manager\BackupConduit , CmGetBackupConduit () , CmSetSystemBackupConduit ()

CmGetSystemConduitByCreator Function

Purpose Retrieves information about a conduit registered for the system, given the conduit's creator ID.

Declared In CondMgr.h

Prototype

```
int CmGetSystemConduitByCreator
    (const char *pCreatorID,
     CmConduitType2 &sConduitInfo)
```

Parameters

→ *pCreatorID*
The creator ID of a conduit, provided as a string.

← *sConduitInfo*
A reference to a [CmConduitType2](#) structure that describes the specified conduit.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_NO_CONDUIT
ERR_NO_MEMORY
ERR_REGISTRY_ACCESS
ERR_INVALID_HANDLE
ERR_INVALID_CREATOR_ID

For descriptions of all Conduit Manager error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments This function retrieves information about a conduit that is registered for the system. To retrieve this information for a conduit that is registered for the current Windows user, call [CmGetConduitByCreator\(\)](#).

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [CmConduitType2](#), [CmInstallSystemConduitByStruct\(\)](#),
[CmGetSystemConduitByIndex\(\)](#),
[CmGetConduitByCreator\(\)](#)

Conduit Manager API

CmGetSystemConduitByIndex

CmGetSystemConduitByIndex Function

Purpose	Retrieves information about a conduit registered for the system given the conduit's index.
Declared In	CondMgr.h
Prototype	<pre>int CmGetSystemConduitByIndex (int iIndex, CmConduitType2 &sConduitInfo)</pre>
Parameters	<p>→ <i>iIndex</i> The zero-based index of a conduit registered for the system. Call CmGetSystemConduitCount() to determine the highest index value.</p> <p>← <i>sConduitInfo</i> A reference to a CmConduitType2 structure that describes the specified conduit.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_INDEX_OUT_OF_RANGE
Comments	This function retrieves information about a conduit that is registered for the system. To retrieve this information for a conduit that is registered for the current Windows user, call CmGetConduitByIndex() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	CmConduitType2 , CmInstallSystemConduitByStruct() , CmGetSystemConduitCount() , CmGetConduitByIndex() , CmGetSystemConduitByCreator()

CmGetSystemConduitCount Function

Purpose	Returns the number of conduits that are registered with HotSync Manager for the system.
Declared In	CondMgr.h
Prototype	<code>int CmGetSystemConduitCount (void)</code>
Parameters	None.
Returns	If successful, returns the number of registered conduits. If unsuccessful, returns the following nonzero error code value: <code>ERR_REGISTRY_ACCESS</code> For descriptions of all Conduit Manager error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	The returned count includes all conduits registered for the system. This count includes conduits registered by placement in the system’s Conduits folder (folder-registered) and those registered by configuration entries (conventionally registered). This count does not include backup or install conduits, nor does it include conduits that are registered for the current Windows user. To retrieve this information for conduits that are registered for the current Windows user, call CmGetConduitCount() . You can use this count to determine the range of valid index values that you can use with CmGetSystemConduitByIndex() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	CmGetConduitCount() , CmGetSystemConduitByIndex()

Conduit Manager API

CmGetSystemConduitCreatorID

CmGetSystemConduitCreatorID Function

Purpose Retrieves the creator ID of a conduit that is registered for the system.

Declared In CondMgr.h

Prototype int CmGetSystemConduitCreatorID (int *iIndex*,
 char **pCreatorID*, int **piSize*)

Parameters → *iIndex*

The zero-based index of a conduit registered for the system.
Call [CmGetSystemConduitCount\(\)](#) to determine the highest index value.

← *pCreatorID*

A character buffer. Upon return, this contains the creator ID string.

↔ *piSize*

A pointer to an integer that specifies the size, in chars, of the buffer referenced by the *pCreatorID* parameter. Upon return, this is the actual size, in chars, of the creator ID string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_REGISTRY_ACCESS

ERR_BUFFER_TOO_SMALL

The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the *piSize* parameter.

For descriptions of all Conduit Manager error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments	Use 0 to specify the first conduit defined in the configuration entries, and increment the index value by 1 for each subsequent conduit.
	This function retrieves information about a conduit that is registered for the system. To retrieve this information for a conduit that is registered for the current Windows user, call <u>CmGetConduitCreatorID()</u> .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	<u>CmGetSystemConduitCount()</u> , <u>CmGetConduitCreatorID()</u> , <u>CmInstallSystemConduitByStruct()</u>

Conduit Manager API

CmGetSystemCreatorDirectory

CmGetSystemCreatorDirectory Function

Purpose Retrieves the name of the directory in which a system-registered conduit stores its user data.

Declared In CondMgr.h

Prototype

```
int CmGetSystemCreatorDirectory
    (const char *pCreatorID, TCHAR *pDirectory,
     int *piSize)
```

Parameters

→ *pCreatorID*
The creator ID string of the conduit whose [Directory](#) value you want to retrieve.

← *pDirectory*
A pointer to a character buffer. Upon return, this is the value of the [Directory](#) configuration entry for the specified conduit.

↔ *piSize*
A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the *pDirectory* parameter. Upon return, this is the actual size, in TCHARS, of the [Directory](#) string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_AMBIGUOUS_CREATORID

ERR_NO_CONDUIT

ERR_REGISTRY_ACCESS

ERR_BUFFER_TOO_SMALL

The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the *piSize* parameter.

ERR_VALUE_NOT_FOUND

ERR_INVALID_CREATOR_ID

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments

This function retrieves the value of the [Directory](#) configuration entry for the specified conduit. The directory string specifies the directory, under the current HotSync user’s directory, that the conduit creates to store its user data and support files. For example, if the value of Directory were DateBook and the value of [Core\Path](#) were a typical value, then the full path to this conduit’s data directory is:

```
C:\Documents and Settings\<WinUsername>\  
My Documents\Palm OS Desktop\  
<HotSyncUsername>\DateBook
```

This function retrieves information about a conduit that is registered for the system. To retrieve this information for a conduit that is registered for the current Windows user, call [CmGetCreatorDirectory\(\)](#).

Compatibility

Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also

[Directory](#), [CmSetSystemCreatorDirectory\(\)](#),
[CmGetCreatorDirectory\(\)](#)

Conduit Manager API

CmGetSystemCreatorFile

CmGetSystemCreatorFile Function

Purpose	Retrieves the name of the data file for a system-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmGetSystemCreatorFile (const char *pCreatorID, TCHAR *pFile, int *piSize)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose File value you want to retrieve.</p> <p>← <i>pFile</i> A pointer to a character buffer. Upon return, this is the value of the File configuration entry for the specified conduit.</p> <p>↔ <i>piSize</i> A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the <i>pFile</i> parameter. Upon return, this is the actual size, in TCHARS, of the File string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_BUFFER_TOO_SMALL The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the <i>piSize</i> parameter.</p> <p>ERR_VALUE_NOT_FOUND ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments	This function retrieves the value of the File configuration entry for the specified conduit. The file string specifies the name of the desktop file to synchronize with the handheld database. This file can be anywhere, but it is commonly located in the directory returned by CmGetSystemCreatorDirectory() . This function retrieves information about a conduit that is registered for the system. To retrieve this information for a conduit that is registered for the current Windows user, call CmGetCreatorFile() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	File , CmSetSystemCreatorFile() , CmGetCreatorFile()

Conduit Manager API

CmGetSystemCreatorIDList

CmGetSystemCreatorIDList Function

Purpose	Retrieves a list of the creator IDs of all conduits registered for the system.
Declared In	CondMgr.h
Prototype	<pre>int CmGetSystemCreatorIDList (CM_CREATORLIST_TYPE *pCreatorList, int *piSize)</pre>
Parameters	<p>→ <i>pCreatorList</i> A pointer to an array of CM_CREATORLIST_ITEM_TYPE structures containing creator IDs.</p> <p>↔ <i>piSize</i> A pointer to an integer value that specifies the number of entries in the array referenced by the <i>pCreatorList</i> parameter. Upon return, this is the actual number of array items that were filled in. If this function fails because the array is too small, the value of <i>piSize</i> upon return is the required number of array items.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_REGISTRY_ACCESS ERR_BUFFER_TOO_SMALL The size of the array that you supplied is too small to contain the number of creator ID items, so this function passes back the required number of items into the <i>piSize</i> parameter.</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments	This function returns the list of creator IDs via a CM_CREATORLIST_TYPE pointer to an array of structures of type CM_CREATORLIST_ITEM_TYPE . This function retrieves information about conduits that are registered for the system. To retrieve this information for conduits that are registered for the current Windows user, call CmGetCreatorIDList() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	CM_CREATORLIST_ITEM_TYPE , CmGetCreatorIDList() , CmInstallSystemConduitByStruct()

Conduit Manager API

CmGetSystemCreatorName

CmGetSystemCreatorName Function

Purpose	Retrieves the filename of a system-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmGetSystemCreatorName (const char *pCreatorID, TCHAR *pConduitName, int *piSize)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Conduit value you want to retrieve.</p> <p>← <i>pConduitName</i> A pointer to a character buffer. Upon return, this is the value of the Conduit configuration entry for the specified conduit.</p> <p>↔ <i>piSize</i> A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the <i>pConduitName</i> parameter. Upon return, this is the actual size, in TCHARS, of the Conduit string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_BUFFER_TOO_SMALL The buffer that you supplied is too small to contain the string, so this function stores the actual required size into the <i>piSize</i> parameter.</p> <p>ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments	This function retrieves the value of the Conduit configuration entry for the specified conduit. This value is the filename of the conduit DLL that is registered with the specified creator ID—for example, ToDoCond.dll is registered with the creator ID 'todo'. This function retrieves information about a conduit that is registered for the system. To retrieve this information for a conduit that is registered for the current Windows user, call CmGetCreatorName () .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	Conduit , CmSetSystemCreatorName () , CmGetCreatorName ()

Conduit Manager API

CmGetSystemCreatorPriority

CmGetSystemCreatorPriority Function

Purpose	Retrieves the execution priority for a system-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmGetSystemCreatorPriority (const char *pCreatorID, DWORD *pdwPriority)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Priority value you want to retrieve.</p> <p>← <i>pdwPriority</i> A pointer to a DWORD value. Upon return, this is the value of the Priority configuration entry for the specified conduit, which is a value between 0 and 4.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER ERR_VALUE_NOT_FOUND</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments	This function retrieves the value of the Priority configuration entry for the specified conduit. HotSync Manager uses the priority value to determine the order in which it executes conduits—that is, the order in which it calls conduits' OpenConduit() entry points. HotSync Manager runs conduits with a value of 0 first and those with 4 last.
	This function retrieves information about a conduit that is registered for the system. To retrieve this information for a conduit that is registered for the current Windows user, call CmGetCreatorPriority() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	Priority , CmSetSystemCreatorPriority() , CmGetCreatorPriority()

Conduit Manager API

CmGetSystemCreatorRemote

CmGetSystemCreatorRemote Function

Purpose	Retrieves the name of a database on the handheld that is associated with a system-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmGetSystemCreatorRemote (const char *pCreatorID, TCHAR *pRemoteDB, int *piSize)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Remote value you want to retrieve.</p> <p>← <i>pRemoteDB</i> A pointer to a character buffer. Upon return, this is the value of the Remote configuration entry for the specified conduit.</p> <p>↔ <i>piSize</i> A pointer to an integer that specifies the size, in TCHARs, of the buffer referenced by the <i>pRemoteDB</i> parameter. Upon return, this is the actual size, in TCHARs, of the Remote string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_BUFFER_TOO_SMALL The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size into the <i>piSize</i> parameter.</p> <p>ERR_VALUE_NOT_FOUND ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER</p>

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments

This function retrieves the value of the [Remote](#) configuration entry for the specified conduit. This value names a single database on the handheld that is associated with this conduit.

NOTE: Though a conduit can access multiple databases on the handheld, `CmGetCreatorRemote()` returns only one handheld database name: the one specified in the `Remote` configuration entry for this conduit. See the description of “[Remote](#)” on page 183 for more information to help you decide whether you want to use this configuration entry.

This function retrieves information about a conduit that is registered for the system. To retrieve this information for a conduit that is registered for the current Windows user, call

[CmGetCreatorRemote\(\)](#).

Compatibility

Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also

[Remote](#), [CmSetSystemCreatorRemote\(\)](#),
[CmGetCreatorRemote\(\)](#)

Conduit Manager API

CmGetSystemCreatorTitle

CmGetSystemCreatorTitle Function

Purpose	Retrieves the display name of a system-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmGetSystemCreatorTitle (const char *pCreatorID, TCHAR *pTitle, int *piSize)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Name value you want to retrieve.</p> <p>← <i>pTitle</i> A pointer to a character buffer. Upon return, this is the value of the Name configuration entry for the specified conduit.</p> <p>↔ <i>piSize</i> A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the <i>pTitle</i> parameter. Upon return, this is the actual size, in TCHARS, of the Name string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_BUFFER_TOO_SMALL The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size into the <i>piSize</i> parameter.</p> <p>ERR_VALUE_NOT_FOUND ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments	This function retrieves the value of the Name configuration entry for the specified conduit. When this value is not NULL, HotSync Manager uses it as the conduit's display name in its Custom and HotSync Progress dialog boxes. If this value is NULL, HotSync Manager uses the value returned by a conduit's GetConduitInfo() entry point, and if that fails, its GetConduitName() entry point. This function retrieves information about a conduit that is registered for the system. To retrieve this information for a conduit that is registered for the current Windows user, call CmGetCreatorTitle() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	Name , CmSetSystemCreatorTitle() , CmGetCreatorTitle()

CmGetSystemCreatorValueDword Function

Purpose	Retrieves a DWORD configuration entry value for a system-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmGetSystemCreatorValueDword (const char *pCreatorID, TCHAR *pValue, DWORD *dwValue, DWORD dwDefault)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit for which you want to retrieve a configuration entry value.</p> <p>→ <i>pValue</i> The configuration entry name.</p> <p>← <i>dwValue</i> Upon return, this contains the value of the configuration entry specified by <i>pValue</i>.</p> <p>→ <i>dwDefault</i> The default value to return in <i>dwValue</i> if the specified <i>pValue</i> name could not be found in the configuration entries.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER ERR_VALUE_NOT_FOUND</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments	This function retrieves a DWORD value from the configuration entries for a conduit. This is a general purpose function for retrieving a conduit configuration entry value by name. You can use it to retrieve values of configuration entries that you create for your own use with CmSetSystemCreatorValueDword() . If Conduit Manager does not find the configuration entry name specified by <i>pValue</i> , it sets <i>dwValue</i> equal to the value of <i>dwDefault</i> and returns 0 (success). However, if the conduit specified by <i>pCreatorID</i> does not exist, this function does <i>not</i> set <i>dwValue</i> equal to the value of <i>dwDefault</i> but does return an error. This function retrieves information about a conduit that is registered for the system. To retrieve this information for a conduit that is registered for the current Windows user, call CmGetCreatorValueDword() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	CmSetSystemCreatorValueDword() , CmGetCreatorValueDword() , CmGetSystemCreatorValueString()

Conduit Manager API

CmGetSystemCreatorValueString

CmGetSystemCreatorValueString Function

Purpose	Retrieves a string configuration entry value for a system-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmGetSystemCreatorValueString (const char *pCreatorID, TCHAR *pValue, TCHAR *pString, int *piSize, TCHAR *pDefault)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit for which you want to retrieve a configuration entry value.</p> <p>→ <i>pValue</i> The configuration entry name.</p> <p>← <i>pString</i> A pointer to a character buffer. Upon return, this is the value of the configuration entry specified by <i>pValue</i>.</p> <p>↔ <i>piSize</i> A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the <i>pString</i> parameter. Upon return, this is the actual size, in TCHARS, of the string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p> <p>→ <i>pDefault</i> The default value to return in <i>pString</i> if the specified <i>pValue</i> name could not be found in the configuration entries.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <ul style="list-style-type: none">ERR_AMBIGUOUS_CREATORIDERR_NO_CONDUITERR_REGISTRY_ACCESSERR_BUFFER_TOO_SMALL<ul style="list-style-type: none">The size of the buffer that you supplied is too small to contain the string, so this function passes back the actual required size into the <i>piSize</i> parameter.

ERR_INVALID_CREATOR_ID

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments

This function retrieves a string value from the configuration entries for a conduit. This is a general purpose function for retrieving a conduit configuration entry value by name. You can use it to retrieve values of configuration entries that you create for your own use with [CmSetSystemCreatorValueString\(\)](#).

This function retrieves information about a conduit that is registered for the system. To retrieve this information for a conduit that is registered for the current Windows user, call [CmGetCreatorValueString\(\)](#).

If Conduit Manager does not find the configuration entry name specified by *pValue*, it passes back the default value (sets *pString* equal to the value of *pDefault*) and returns 0 (success). However, if the conduit specified by *pCreatorID* does not exist, Conduit Manager does *not* pass back the default value but does return an error.

Compatibility

Conduit Manager versions: 3 and later.

Palm OS versions: All.

See Also

[CmSetSystemCreatorValueString\(\)](#),
[CmGetCreatorValueString\(\)](#),
[CmGetSystemCreatorValueDword\(\)](#)

Conduit Manager API

CmGetSystemDiscoveryInfoByIndex

CmGetSystemDiscoveryInfoByIndex Function

Purpose	Retrieves information about how HotSync Manager discovers a system-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmGetSystemDiscoveryInfoByIndex (int iIndex, CmDiscoveryInfoType &sDiscoveryInfo)</pre>
Parameters	<p>→ <i>iIndex</i> The zero-based index of a conduit registered for the system. Call CmGetSystemConduitCount() to determine the highest index value.</p> <p>← <i>sDiscoveryInfo</i> A reference to a CmDiscoveryInfoType structure that describes whether HotSync Manager can discover the specified conduit and if so, whether the conduit is in a Conduits folder or in the conduit configuration entries.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_INDEX_OUT_OF_RANGE
Comments	Note that if the bLoadable field of the CmDiscoveryInfoType structure is FALSE, then the specified conduit is not properly registered and HotSync Manager cannot call it. This function retrieves information about a conduit that is registered for the system. To retrieve this information for a conduit that is registered for the current Windows user, call CmGetDiscoveryInfoByIndex() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	CmDiscoveryInfoType , CmGetDiscoveryInfoByIndex()

CmGetSystemHotSyncExecPath Function

Purpose Retrieves the path and filename of the HotSync Manager executable on the desktop computer. (Strip off the HotSync .exe filename to get the path of support DLLs.)

Declared In CondMgr.h

Prototype int CmGetSystemHotSyncExecPath (TCHAR **pPath*,
 int **piSize*)

Parameters ← *pPath*
A pointer to a character buffer. Upon return, this is the path and filename of the HotSync Manager executable—for example, C:\Program Files\PalmSource\Desktop\HotSync.exe.

↔ *piSize*
A pointer to an integer that specifies the size, in TCHARs, of the buffer referenced by the *pPath* parameter. Upon return, this is the actual size, in TCHARs, of the string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_REGISTRY_ACCESS

ERR_BUFFER_TOO_SMALL

The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size into the *piSize* parameter.

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Conduit Manager API

CmGetSystemHotSyncExecPath

Comments This function retrieves the value of the [Core\HotSyncPath](#) configuration entry for HotSync Manager. This value is the path and filename of the HotSync Manager executable on the desktop computer. If HotSync Manager is not installed, the *pPath* parameter points to an empty string.

NOTE: To determine the path of support DLLs like CondMgr.dll, Instaide.dll and UserData.dll, call this function and strip off the filename HotSync.exe from the end of the character buffer pointed to by *pPath*.

HotSync Manager Versions Earlier than 6

HotSync Manager versions *earlier than 6* set this path only when they are launched with the -r or -d command-line options, which usually happens only when HotSync Manager is installed.

HotSync Manager Versions 6 and Later

HotSync Manager versions *6 and later* actively set this path every time HotSync Manager is launched. Setting this path on every launch enables developers to switch more easily between using the HotSync Manager that ships with the CDK and the one that ships with Palm OS® Desktop software. Whichever HotSync.exe you launch sets this path to its current directory. Because only one HotSync Manager executable is used by all Windows users, it would seem that setting this path for the system (local machine) is all that is necessary. However, HotSync Manager actually sets this path for both the current Windows user and for the system. This redundancy is to support legacy conduit installers from third party developers that use this path for the current Windows user to install their conduits in the HotSync Manager executable's directory. Therefore for consistency, the Conduit Manager API includes two functions:

- [CmGetHotSyncExecPath\(\)](#) retrieves the path stored for the current Windows user (Conduit Manager versions 2 and later). This function is deprecated as of CDK 6.0, but it continues to be available to support legacy installers.
- [CmGetSystemHotSyncExecPath\(\)](#) retrieves the path stored for the system (Conduit Manager versions 3 and later).

These paths are stored separately, but HotSync Manager always sets both values every time it is launched. The only exception is when a Windows user with no administrator privileges runs HotSync Manager. In this case, HotSync Manager cannot set this path for the system, and no system-level Conduit Manager API can read or write system-level configuration entries.

NOTE: PalmSource recommends that you use the [CmGetSystemHotSyncExecPath\(\)](#) function, because [CmGetHotSyncExecPath\(\)](#) may be obsoleted in future versions of Conduit Manager.

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [CmGetCorePath\(\)](#), [CmGetHotSyncExecPath\(\)](#)

Conduit Manager API

CmInstallConduit

CmInstallConduit Function

Purpose	Registers a conduit with HotSync Manager for the current Windows user. Uses a CmConduitType structure.
Declared In	CondMgr.h
Prototype	<code>int CmInstallConduit (HANDLE <i>hStruct</i>)</code>
Parameters	<p><code>→ <i>hStruct</i></code> The handle of the conduit to install. This is a handle to a CmConduitType structure.</p>
Returns	<p>If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_CONDUIT_MGR ERR_NO_CONDUIT ERR_NO_MEMORY ERR_CREATORID_ALREADY_IN_USE ERR_REGISTRY_ACCESS ERR_UNABLE_TO_CREATE_CONDUIT ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_HANDLE ERR_INVALID_CREATOR_ID</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments To register a conduit, fill in the fields of a [CmConduitType](#) structure and pass this function its handle.

NOTE: PalmSource recommends that you use [CmInstallConduitByStruct\(\)](#) or [CmInstallSystemConduitByStruct\(\)](#) instead. These functions do not require the kind of memory management that `CmInstallConduit()` does. For more on registering a conduit conventionally with a single structure, see “[Writing a Single Structure](#)” on page 106 in the *C/C++ Sync Suite Companion*.

This function registers a conduit for the current Windows user. To register a conduit for the system, call [CmInstallSystemConduitByStruct\(\)](#) or [CmInstallSystemCreator\(\)](#).

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [CmConduitType](#), [CmInstallConduitByStruct\(\)](#),
[CmInstallCreator\(\)](#),
[CmInstallSystemConduitByStruct\(\)](#),
[CmInstallSystemCreator\(\)](#)

Conduit Manager API

CmInstallConduitByStruct

CmInstallConduitByStruct Function

Purpose	Registers a conduit with HotSync Manager for the current Windows user. Uses a CmConduitType2 structure.
Declared In	CondMgr.h
Prototype	<pre>int CmInstallConduitByStruct (CmConduitType2 &sConduitInfo)</pre>
Parameters	<p>→ <i>sConduitInfo</i> A reference to a CmConduitType2 structure that specifies the conduit registration information.</p>
Returns	<p>If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_CONDUIT_MGR ERR_NO_CONDUIT ERR_NO_MEMORY ERR_CREATORID_ALREADY_IN_USE ERR_REGISTRY_ACCESS ERR_UNABLE_TO_CREATE_CONDUIT ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_CREATOR_ID</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments	To register a conduit, fill in the fields of a CmConduitType2 structure and pass this function a reference to it. For more on registering a conduit conventionally with a single structure, see “ Writing a Single Structure ” on page 106 in the <i>C/C++ Sync Suite Companion</i> . This function registers a conduit for the current Windows user. To register a conduit for the system, call CmInstallSystemConduitByStruct() or CmInstallSystemCreator() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	CmConduitType2 , CmInstallCreator() , CmInstallSystemConduitByStruct() , CmInstallSystemCreator()

Conduit Manager API

CmInstallCreator

CmInstallCreator Function

Purpose Registers a new conduit creator ID with HotSync Manager for the current Windows user (requires calling other CmSetCreator...() functions to complete registration).

Declared In CondMgr.h

Prototype int CmInstallCreator (const char **pCreator*,
 int *iType*)

Parameters → *pCreator*

The creator ID string for the conduit. This sets the [Creator](#) configuration entry.

→ *iType*

The conduit type. This is one of the values described in “[Conduit Information Types](#)” on page 661. For almost all conduits, specify the CONDUIT_APPLICATION value.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_CREATORID_ALREADY_IN_USE

ERR_REGISTRY_ACCESS

ERR_UNABLE_TO_CREATE_CONDUIT

ERR_UNABLE_TO_SET_CONDUIT_VALUE

ERR_INVALID_CREATOR_ID

ERR_INVALID_CREATOR_TYPE

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments	<p>This function creates and sets the Creator value for a new set of conduit configuration entries. To use this function to register a conduit, you must call this function first; then if this function call succeeds, call other <code>CmSetCreator...()</code> functions to specify the conduit's remaining configuration entries.</p> <p>Though HotSync Manager does not distinguish between values you pass in the <i>iType</i> parameter, you must still pass in a value, which for almost all conduits is <code>CONDUIT_APPLICATION</code>.</p> <p>This function sets the creator ID of a conduit that is not yet registered for the current Windows user. To set this value for a conduit that is not yet registered for the system, call CmInstallSystemCreator() or CmInstallSystemConduitByStruct().</p>
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	Creator , CmInstallConduitByStruct() , CmInstallSystemCreator() , CmInstallSystemConduitByStruct()

Conduit Manager API

CmInstallSystemConduitByStruct

CmInstallSystemConduitByStruct Function

Purpose	Registers a conduit with HotSync Manager for the system. Uses a CmConduitType2 structure.
Declared In	CondMgr.h
Prototype	<pre>int CmInstallSystemConduitByStruct (const CmConduitType2 &sConduitInfo)</pre>
Parameters	<p>→ <i>sConduitInfo</i> A reference to a CmConduitType2 structure that specifies the conduit registration information.</p>
Returns	<p>If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_CONDUIT_MGR ERR_NO_CONDUIT ERR_NO_MEMORY ERR_CREATORID_ALREADY_IN_USE ERR_REGISTRY_ACCESS ERR_UNABLE_TO_CREATE_CONDUIT ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_CREATOR_ID ERR_INSUFFICIENT_PRIVILEGES The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required.</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments	To register a conduit, fill in the fields of a CmConduitType2 structure and pass this function a reference to it. For more on registering a conduit conventionally with a single structure, see “ Writing a Single Structure ” on page 106 in the <i>C/C++ Sync Suite Companion</i> . This function registers a conduit for the system. To register a conduit for the current Windows user, call CmInstallConduitByStruct() or CmInstallConduit() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	CmConduitType2 , CmInstallSystemCreator() , CmInstallConduitByStruct() , CmInstallCreator()

Conduit Manager API

CmInstallSystemCreator

CmInstallSystemCreator Function

Purpose	Registers a new conduit creator ID with HotSync Manager for system (requires calling other CmSetSystemCreator...() functions to complete registration).
Declared In	CondMgr.h
Prototype	<pre>int CmInstallSystemCreator (const char *pCreator, int iType)</pre>
Parameters	<p>→ <i>pCreator</i> The creator ID string for the conduit. This sets the Creator configuration entry.</p> <p>→ <i>iType</i> The conduit type. This is one of the values described in "Conduit Information Types" on page 661. For almost all conduits, specify the CONDUIT_APPLICATION value.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_CREATORID_ALREADY_IN_USE ERR_REGISTRY_ACCESS ERR_UNABLE_TO_CREATE_CONDUIT ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_CREATOR_ID ERR_INVALID_CREATOR_TYPE ERR_INSUFFICIENT_PRIVILEGES The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required.</p> <p>For more information about the error codes, see "Conduit Manager Error Codes" on page 843.</p>

Comments	This function creates and sets the Creator value for a new set of conduit configuration entries. To use this function to register a conduit, you must call this function first; then if this function call succeeds, call other <code>CmSetSystemCreator...()</code> functions to specify the conduit's remaining configuration entries. Though HotSync Manager does not distinguish between values you pass in the <i>iType</i> parameter, you must still pass in a value, which for almost all conduits is CONDUIT_APPLICATION. This function sets the creator ID of a conduit that is not yet registered for the system. To set this value for a conduit that is not yet registered for the current Windows user, call CmInstallCreator() or CmInstallConduitByStruct() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	Creator , CmInstallSystemConduitByStruct() , CmInstallCreator() , CmInstallConduitByStruct()

Conduit Manager API

CmIsCurrentUserAdmin

CmIsCurrentUserAdmin Function

Purpose	Determines whether the current user has Windows administrator privileges.
Declared In	CondMgr.h
Prototype	BOOL CmIsCurrentUserAdmin ()
Parameters	None.
Returns	If the current user has administrator privileges, returns TRUE. If not, returns FALSE.
Comments	All Conduit Manager functions that set a configuration entry for the system (CmSet...System...() functions and others) return an ERR_INSUFFICIENT_PRIVILEGES error if the current Windows user does not have administrator privileges, but only on an NTFS file system. To determine whether these functions will fail before you call them, call CmIsCurrentUserAdmin() first. For a user without administrator privileges, Conduit Manager functions can still read system-level information, though.
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.

CmRemoveConduitByCreatorID Function

Purpose Unregisters a conduit that is registered for the current Windows user, given the conduit's creator ID.

Declared In CondMgr.h

Prototype

```
int CmRemoveConduitByCreatorID  
    (const char *pCreatorID)
```

Parameters

→ *pCreatorID*
The creator ID string for a conduit.

Returns If successful, returns the number of conduits that were removed. This value is always 1.

If unsuccessful, returns one of the following nonzero error code values:

ERR_FOLDER_NOT_FOUND

The specified folder-registered conduit cannot be disabled because the Disabled folder does not exist and cannot be created.

ERR_UNABLE_TO_DELETE

ERR_NO_CONDUIT

ERR_REGISTRY_ACCESS

ERR_INVALID_CREATOR_ID

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments Given its creator ID, this function unregisters a conduit. If the conduit was conventionally registered—for example, using [CmInstallConduitByStruct\(\)](#)—then this function removes all of its configuration entries. However, if the conduit was registered by being placed in the Conduits folder of the current Windows user, then this function moves the conduit to the Disabled folder. Therefore for folder-registered conduits, calling this function has the same result as calling

[FmDisableCurrentUserConduitByPath\(\)](#).

This function unregisters a conduit for the current Windows user. To unregister a conduit for the system, call [CmRemoveSystemConduitByCreatorID\(\)](#) or [CmRemoveSystemConduitByIndex\(\)](#).

Conduit Manager API

CmRemoveConduitByCreatorID

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [CmRemoveConduitByIndex\(\)](#),
[CmRemoveSystemConduitByCreatorID\(\)](#),
[CmRemoveSystemConduitByIndex\(\)](#)

CmRemoveConduitByIndex Function

Purpose Unregisters a conduit that is registered for the current Windows user, given the conduit's index.

Declared In CondMgr.h

Prototype int CmRemoveConduitByIndex (int *iIndex*)

Parameters → *iIndex*

The zero-based index of a conduit registered for the current Windows user. Call [CmGetConduitCount\(\)](#) to determine the highest index value.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_FOLDER_NOT_FOUND

The specified folder-registered conduit cannot be disabled because the Disabled folder does not exist and cannot be created.

ERR_INDEX_OUT_OF_RANGE

ERR_UNABLE_TO_DELETE

ERR_REGISTRY_ACCESS

Comments Given a zero-based index, this function unregisters a conduit. If the conduit was conventionally registered—for example, using [CmInstallConduitByStruct\(\)](#)—then this function removes all of its configuration entries. However, if the conduit was registered by being placed in the Conduits folder of the current Windows user, then this function moves the conduit to the Disabled folder. Therefore for folder-registered conduits, calling this function has the same result as calling [FmDisableCurrentUserConduitByPath\(\)](#).

This function unregisters a conduit for the current Windows user. To unregister a conduit for the system, call [CmRemoveSystemConduitByCreatorID\(\)](#) or [CmRemoveSystemConduitByIndex\(\)](#).

Conduit Manager API

CmRemoveConduitByIndex

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [CmRemoveConduitByCreatorID\(\)](#),
[CmRemoveSystemConduitByIndex\(\)](#),
[CmRemoveSystemConduitByCreatorID\(\)](#)

CmRemoveSystemConduitByCreatorID Function

Purpose Unregisters a conduit that is registered for the system, given the conduit's creator ID.

Declared In CondMgr.h

Prototype int CmRemoveSystemConduitByCreatorID
(const char **pCreatorID*)

Parameters → *pCreatorID*
The creator ID string for a conduit.

Returns If successful, returns the number of conduits that were removed. This value is always 1.

If unsuccessful, returns one of the following nonzero error code values:

ERR_FOLDER_NOT_FOUND

The specified folder-registered conduit cannot be disabled because the Disabled folder does not exist and cannot be created.

ERR_UNABLE_TO_DELETE

ERR_NO_CONDUIT

ERR_REGISTRY_ACCESS

ERR_INVALID_CREATOR_ID

ERR_INSUFFICIENT_PRIVILEGES

The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required.

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Conduit Manager API

CmRemoveSystemConduitByCreatorID

Comments	Given its creator ID, this function unregisters a conduit. If the conduit was conventionally registered—for example, using CmInstallSystemConduitByStruct() —then this function removes all of its configuration entries. However, if the conduit was registered by being placed in the system's Conduits folder, then this function moves the conduit to the system's Disabled folder. Therefore for folder-registered conduits, calling this function has the same result as calling FmDisableSystemConduitByPath() . This function unregisters a conduit for the system. To unregister a conduit for the current Windows user, call CmRemoveConduitByCreatorID() or CmRemoveConduitByIndex() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	CmRemoveSystemConduitByIndex() , CmRemoveConduitByCreatorID() , CmRemoveConduitByIndex()

CmRemoveSystemConduitByIndex Function

Purpose Unregisters a conduit that is registered for the system, given the conduit's index.

Declared In CondMgr.h

Prototype int CmRemoveSystemConduitByIndex (int *iIndex*)

Parameters → *iIndex*

The zero-based index of a conduit registered for the current Windows user. Call [CmGetSystemConduitCount\(\)](#) to determine the highest index value.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_FOLDER_NOT_FOUND

The specified folder-registered conduit cannot be disabled because the Disabled folder does not exist and cannot be created.

ERR_INDEX_OUT_OF_RANGE

ERR_UNABLE_TO_DELETE

ERR_REGISTRY_ACCESS

ERR_INSUFFICIENT_PRIVILEGES

The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required.

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments Given a zero-based index, this function unregisters a conduit. If the conduit was conventionally registered—for example, using [CmInstallSystemConduitByStruct\(\)](#)—then this function removes all of its configuration entries. However, if the conduit was registered by being placed in the Conduits folder of the current Windows user, then this function moves the conduit to the Disabled folder. Therefore for folder-registered conduits, calling this function has the same result as calling [FmDisableSystemConduitByPath\(\)](#).

Conduit Manager API

CmRemoveSystemConduitByIndex

This function unregisters a conduit for the system. To unregister a conduit for the current Windows user, call [CmRemoveConduitByIndex\(\)](#) or [CmRemoveConduitByCreatorID\(\)](#).

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [CmRemoveSystemConduitByCreatorID\(\)](#),
[CmRemoveConduitByIndex\(\)](#),
[CmRemoveConduitByCreatorID\(\)](#)

CmRestoreHotSyncSettings Function

Purpose (*Degraded*) Restores HotSync Manager configuration entry settings to their default values.

Declared In CondMgr.h

Prototype int CmRestoreHotSyncSettings (BOOL *bToDefaults*)

Parameters → *bToDefaults*

If TRUE, remove and re-add entries using default settings. If FALSE, restore entries to default values, except for any settings that have been overwritten.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_FOLDER_NOT_FOUND

The Conduits or Disabled folder does not exist and cannot be created.

ERR_INSUFFICIENT_PRIVILEGES

The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required.

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments As of CDK 6.0, this function is deprecated and will be obsoleted in a future version of the Conduit Manager API.

WARNING! Do not use this function. Calling it can cause some third-party conduits to fail.

This function restores the [Core\HotSyncPath](#) and [Core\Path](#) configuration entries for HotSync Manager to their default values. It also restores the configuration entries for the default conduits (Address Book, Date Book, To Do, Memo Pad, Backup, and install conduits) and notifiers.

If the *bToDefaults* parameter is TRUE, then this function first removes these settings and then adds them again using default values. This means that the default conduits and other settings are restored, but no other conduit’s configuration entries are modified.

Conduit Manager API

CmRestoreHotSyncSettings

If *bToDefaults* is FALSE, then this function resets these settings to their default values, but no configuration entries that have been overwritten by third-party settings are reset. For example, if a third-party conduit is registered with the creator ID 'date', then calling this function with *bToDefaults* = TRUE overwrites that conduit's configuration entries, but with *bToDefaults* = FALSE it does not.

Compatibility

Conduit Manager versions: 2 and later.
Palm OS versions: All.

CmSetBackupConduit Function

Purpose	Sets the path or filename of the backup conduit for the current Windows user.
Declared In	CondMgr.h
Prototype	int CmSetBackupConduit (const TCHAR * <i>pConduit</i>)
Parameters	<i>→ pConduit</i> The full path or filename of the conduit used as the backup conduit.
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_REGISTRY_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_POINTER For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	This function establishes the value of the HotSync Manager\BackupConduit entry used by HotSync Manager in the configuration entries. If you specify a filename that Conduit Manager cannot find (or a NULL filename), this function leaves the filename unchanged and returns no error. Before calling this function, you must check that the file exists where Conduit Manager can find it. This function sets the backup conduit for the current Windows user. To set the backup conduit for the system, call CmSetSystemBackupConduit() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	HotSync Manager\BackupConduit , CmGetBackupConduit() , CmSetSystemBackupConduit()

Conduit Manager API

CmSetComPort

CmSetComPort Function

Purpose Sets the name of the COM port that HotSync Manager uses for the specified type of connection on the desktop computer.

Declared In CondMgr.h

Prototype int CmSetComPort (int *iType*, const TCHAR **pPort*)

Parameters

→ *iType*
The type of connection for which to set the COM port. You must specify one of the values described in “[Communications Ports](#)” on page 662.

→ *pPort*
A pointer to a character buffer that specifies the port name—for example, “COM1”.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_REGISTRY_ACCESS
ERR_UNABLE_TO_SET_CONDUIT_VALUE
ERR_INVALID_POINTER
ERR_INVALID_COM_PORT_TYPE

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments This function sets either the [HotSync Manager\ModemComPort](#) or [HotSync Manager\DirectComPort](#) entry from the HotSync Manager portion of the configuration entries, depending on the value you specify for the *iType* parameter. Note that these values are set for the current Windows user. There are no corresponding settings for the system only. HotSync Manager sets default values for each new Windows users.

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [HotSync Manager\ModemComPort](#), [HotSync Manager\DirectComPort](#), [CmGetComPort\(\)](#)

CmSetCorePath Function

Purpose (*Deprecated*) Sets the path of the HotSync users directory for the current Windows user.

Declared In CondMgr.h

Prototype int CmSetCorePath (const TCHAR **pPath*)

Parameters → *pPath*
A pointer to a character buffer that contains the path.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_COULD_NOT_CREATE_DIRECTORY

The specified directory could not be created.

ERR_REGISTRY_ACCESS

ERR_UNABLE_TO_SET_CONDUIT_VALUE

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments As of CDK 6.0, this function is deprecated and will be obsoleted in a future version of the Conduit Manager API.

WARNING! Do not use this function. Changing this path could result in conduits failing to find their desktop data files. Use [CmGetCorePath\(\)](#) to retrieve this path, but do not change it.

This function sets the value of the [Core\Path](#) configuration entry for the current Windows user. Because this path may differ for each Windows user, there is no corresponding “system-level” path.

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [CmGetCorePath\(\)](#), [CmGetHotSyncExecPath\(\)](#)

Conduit Manager API

CmSetCreatorArgument

CmSetCreatorArgument Function

Purpose	(<i>Degraded</i>) Sets the value of the Arguments configuration entry for the specified conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmSetCreatorArgument (const char *pCreatorID, const TCHAR *pArgument)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Arguments value you want to set.</p> <p>→ <i>pArgument</i> The string value that you want to set for the Arguments configuration entry for the specified conduit.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER ERR_CONDUIT_READ_ONLY The specified conduit is folder-registered, therefore its registration information is read-only. For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	As of CDK 6.0, this function is deprecated. No version of HotSync Manager uses this value.
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	Arguments , CmGetCreatorArgument ()

CmSetCreatorDirectory Function

Purpose Sets the name of the directory in which a user-registered conduit stores its user data.

Declared In CondMgr.h

Prototype int CmSetCreatorDirectory (const char **pCreatorID*, const TCHAR **pDirectory*)

Parameters

- *pCreatorID*
The creator ID string of the conduit whose [Directory](#) value you want to set.
- *pDirectory*
The string value that you want to set for the [Directory](#) configuration entry for the specified conduit.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

- ERR_AMBIGUOUS_CREATORID
- ERR_NO_CONDUIT
- ERR_REGISTRY_ACCESS
- ERR_UNABLE_TO_SET_CONDUIT_VALUE
- ERR_INVALID_CREATOR_ID
- ERR_INVALID_POINTER
- ERR_CONDUIT_READ_ONLY

The specified conduit is folder-registered, therefore its registration information is read-only.

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Conduit Manager API

CmSetCreatorDirectory

Comments This function sets the value of the [Directory](#) configuration entry for the specified conduit. The directory string specifies the directory, under the current HotSync user's directory, that the conduit creates to store its user data and support files. For example, if the value of Directory were DateBook and the value of [Core\Path](#) were a typical value, then the full path to this conduit's data directory is:

```
C:\Documents and Settings\<WinUsername>\  
My Documents\Palm OS Desktop\  
<HotSyncUsername>\DateBook
```

This function sets information for a conduit that is registered for the current Windows user. To set this information for a conduit that is registered for the system, call [CmSetSystemCreatorDirectory\(\)](#).

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [Directory](#), [CmGetCreatorDirectory\(\)](#),
[CmSetSystemCreatorDirectory\(\)](#)

CmSetCreatorFile Function

Purpose	Sets the name of the data file for a user-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmSetCreatorFile (const char *pCreatorID, const TCHAR *pFile)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose File value you want to set.</p> <p>→ <i>pFile</i> The string value that you want to set for the File configuration entry for the specified conduit.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER ERR_CONDUIT_READ_ONLY The specified conduit is folder-registered, therefore its registration information is read-only.</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Conduit Manager API

CmSetCreatorFile

Comments	This function sets the value of the File configuration entry for the specified conduit. The file string specifies the name of the desktop file to synchronize with the handheld database. This file can be anywhere, but it is commonly located in the directory returned by CmGetCreatorDirectory() . This function sets information for a conduit that is registered for the current Windows user. To set this information for a conduit that is registered for the system, call CmSetSystemCreatorFile() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	File , CmGetCreatorFile() , CmSetSystemCreatorFile()

CmSetCreatorInfo Function

Purpose (*Deprecated*) Sets a string associated with a user-registered conduit.

Declared In CondMgr.h

Prototype int CmSetCreatorInfo (const char **pCreatorID*,
 const TCHAR **pInfo*)

Parameters → *pCreatorID*

The creator ID string of the conduit whose [Information](#) value you want to set.

→ *pInfo*

The string value that you want to set for the [Information](#) configuration entry for the specified conduit.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_AMBIGUOUS_CREATORID

ERR_NO_CONDUIT

ERR_REGISTRY_ACCESS

ERR_UNABLE_TO_SET_CONDUIT_VALUE

ERR_INVALID_CREATOR_ID

ERR_INVALID_POINTER

ERR_CONDUIT_READ_ONLY

The specified conduit is folder-registered, therefore its registration information is read-only.

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments As of CDK 6.0, this function is deprecated. No version of HotSync Manager uses this value.

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [Information](#), [CmGetCreatorInfo\(\)](#)

Conduit Manager API

CmSetCreatorIntegrate

CmSetCreatorIntegrate Function

Purpose	(<i>Degraded</i>) Sets the value of the Integrate configuration entry for the specified conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmSetCreatorIntegrate (const char *pCreatorID, DWORD dwIntegrate)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Integrate value you want to set.</p> <p>→ <i>dwIntegrate</i> The value that you want to set for the Integrate configuration entry for the specified conduit.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER ERR_CONDUIT_READ_ONLY The specified conduit is folder-registered, therefore its registration information is read-only. For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	As of CDK 6.0, this function is deprecated. No version of HotSync Manager uses this value.
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	Integrate , CmGetCreatorIntegrate()

CmSetCreatorModule Function

Purpose (*Deprecated*) Sets the value of the [Module](#) configuration entry for the specified conduit.

Declared In CondMgr.h

Prototype

```
int CmSetCreatorModule (const char *pCreatorID,  
                      const TCHAR *pModule)
```

Parameters

→ *pCreatorID*

The creator ID string of the conduit whose [Module](#) value you want to set.

→ *pModule*

The string value that you want to set for the [Module](#) configuration entry for the specified conduit.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_AMBIGUOUS_CREATORID

ERR_NO_CONDUIT

ERR_REGISTRY_ACCESS

ERR_UNABLE_TO_SET_CONDUIT_VALUE

ERR_INVALID_CREATOR_ID

ERR_INVALID_POINTER

ERR_CONDUIT_READ_ONLY

The specified conduit is folder-registered, therefore its registration information is read-only.

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments As of CDK 6.0, this function is deprecated. No version of HotSync Manager uses this value.

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [Module](#), [CmGetCreatorModule\(\)](#)

Conduit Manager API

CmSetCreatorName

CmSetCreatorName Function

Purpose Sets the filename of a user-registered conduit.

Declared In CondMgr.h

Prototype int CmSetCreatorName (const char **pCreatorID*,
 const TCHAR **pConduitName*)

Parameters

- *pCreatorID*
The creator ID string of the conduit whose [Conduit](#) value you want to set.
- *pConduitName*
The string value that you want to set for the [Conduit](#) configuration entry for the specified conduit.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_AMBIGUOUS_CREATORID

ERR_NO_CONDUIT

ERR_REGISTRY_ACCESS

ERR_UNABLE_TO_SET_CONDUIT_VALUE

ERR_INVALID_CREATOR_ID

ERR_INVALID_POINTER

ERR_CONDUIT_READ_ONLY

The specified conduit is folder-registered, therefore its registration information is read-only.

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments	This function sets the value of the Conduit configuration entry for the specified conduit. This value is the filename of the conduit DLL that is registered with the specified creator ID—for example, conduit filename ToDoCond.dll is registered with the creator ID 'todo'. This function sets information for a conduit that is registered for the current Windows user. To set this information for a conduit that is registered for the system, call CmSetSystemCreatorName() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	Conduit , CmGetCreatorName() , CmSetSystemCreatorName()

Conduit Manager API

CmSetCreatorPriority

CmSetCreatorPriority Function

Purpose	Sets the execution priority for a user-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmSetCreatorPriority (const char *pCreatorID, DWORD dwPriority)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Priority value you want to set.</p> <p>→ <i>dwPriority</i> The value that you want to set for the Priority configuration entry for the specified conduit. This value must be in the range 0 to 4.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER ERR_CONDUIT_READ_ONLY The specified conduit is folder-registered, therefore its registration information is read-only. For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.

Comments	This function retrieves the value of the Priority configuration entry for the specified conduit. HotSync Manager uses the priority value to determine the order in which it executes conduits—that is, the order in which it calls conduits' OpenConduit() entry points. HotSync Manager runs conduits with a value of 0 first and those with 4 last.
	This function sets information for a conduit that is registered for the current Windows user. To set this information for a conduit that is registered for the system, call CmSetSystemCreatorPriority() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	Priority , CmGetCreatorPriority() , CmSetSystemCreatorPriority()

Conduit Manager API

CmSetCreatorRemote

CmSetCreatorRemote Function

Purpose	Sets the name of a database on the handheld that is associated with a user-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmSetCreatorRemote (const char *pCreatorID, const TCHAR *pRemoteDB)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Remote value you want to set.</p> <p>→ <i>pRemoteDB</i> The string value that you want to set for the Remote configuration entry for the specified conduit.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER ERR_CONDUIT_READ_ONLY The specified conduit is folder-registered, therefore its registration information is read-only. For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.

Comments	This function sets the value of the Remote configuration entry for the specified conduit. This value names a single database on the handheld that is associated with this conduit.
	NOTE: Though a conduit can access multiple databases on the handheld, <code>CmSetCreatorRemote()</code> sets only one handheld database name, which is stored in the <code>Remote</code> configuration entry for this conduit. See the description of “ Remote ” on page 183 for more information to help you decide whether you want to use this configuration entry.
	This function sets information for a conduit that is registered for the current Windows user. To set this information for a conduit that is registered for the system, call CmSetSystemCreatorRemote() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	Remote , CmGetCreatorRemote() , CmSetSystemCreatorRemote()

Conduit Manager API

CmSetCreatorTitle

CmSetCreatorTitle Function

Purpose	Sets the display name of a user-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmSetCreatorTitle (const char *pCreatorID, const TCHAR *pTitle)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Name value you want to set.</p> <p>→ <i>pTitle</i> The string value that you want to set for the Name configuration entry for the specified conduit.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER ERR_CONDUIT_READ_ONLY The specified conduit is folder-registered, therefore its registration information is read-only.</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments	This function sets the value of the Name configuration entry for the specified conduit. When this value is not NULL, HotSync Manager uses it as the conduit's display name in its Custom and HotSync Progress dialog boxes. If this value is NULL, HotSync Manager uses the value returned by a conduit's GetConduitInfo() entry point, and if that fails, its GetConduitName() entry point.
Compatibility	This function sets information for a conduit that is registered for the current Windows user. To set this information for a conduit that is registered for the system, call CmSetSystemCreatorTitle() .
See Also	Name , CmGetCreatorTitle() , CmSetSystemCreatorTitle()

Conduit Manager API

CmSetCreatorUser

CmSetCreatorUser Function

Purpose	(<i>Degraded</i>) Sets the value of the Username configuration entry for the specified conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmSetCreatorUser (const char *pCreatorID, const TCHAR *pUsername)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Username value you want to set.</p> <p>→ <i>pUsername</i> The string value that you want to set for the Username configuration entry for the specified conduit.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER ERR_CONDUIT_READ_ONLY The specified conduit is folder-registered, therefore its registration information is read-only. For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	As of CDK 6.0, this function is deprecated. No version of HotSync Manager uses this value.
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	Username , CmGetCreatorUser()

CmSetCreatorValueDword Function

Purpose Sets the value of a DWORD configuration entry for a user-registered conduit.

Declared In CondMgr.h

Prototype

```
int CmSetCreatorValueDword
    (const char *pCreatorID, TCHAR *pValue,
     DWORD dwValue)
```

Parameters

→ *pCreatorID*
The creator ID string of the conduit whose value you want to set.

→ *pValue*
The configuration entry name.

→ *dwValue*
The value that you want established for the specified configuration entry for the specified conduit.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_AMBIGUOUS_CREATORID

ERR_NO_CONDUIT

ERR_REGISTRY_ACCESS

ERR_INVALID_CREATOR_ID

ERR_INVALID_POINTER

ERR_VALUE_NOT_FOUND

ERR_CONDUIT_READ_ONLY

The specified conduit is folder-registered, therefore its registration information is read-only.

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Conduit Manager API

CmSetCreatorValueDword

Comments	This function sets a DWORD value in the configuration entries for a conduit. This is a general purpose function for setting a conduit configuration entry value by name. You can use it to set values of configuration entries that you retrieve for your own use with CmGetCreatorValueDword() . This function sets information for a conduit that is registered for the current Windows user. To set this information for a conduit that is registered for the system, call CmSetSystemCreatorValueDword() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	CmGetCreatorValueDword() , CmSetSystemCreatorValueDword() , CmSetCreatorValueString()

CmSetCreatorValueString Function

Purpose Sets the value of a string configuration entry for a user-registered conduit

Declared In CondMgr.h

Prototype

```
int CmSetCreatorValueString
    (const char *pCreatorID, TCHAR *pValue,
     TCHAR *pString)
```

Parameters

- *pCreatorID*
The creator ID string of the conduit whose value you want to set.
- *pValue*
The configuration entry name.
- *pString*
The string value that you want established for the specified configuration entry for the specified conduit.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_AMBIGUOUS_CREATORID

ERR_NO_CONDUIT

ERR_REGISTRY_ACCESS

ERR_INVALID_CREATOR_ID

ERR_INVALID_POINTER

ERR_VALUE_NOT_FOUND

ERR_CONDUIT_READ_ONLY

The specified conduit is folder-registered, therefore its registration information is read-only.

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Conduit Manager API

CmSetValueString

Comments	This function sets a string value in the configuration entries for a conduit. This is a general purpose function for setting a conduit configuration entry value by name. You can use it to set values of configuration entries that you retrieve for your own use with CmGetCreatorValueString() . This function sets information for a conduit that is registered for the current Windows user. To set this information for a conduit that is registered for the system, call CmSetSystemCreatorValueString() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	CmGetCreatorValueString() , CmSetSystemCreatorValueString() , CmSetCreatorValueDword()

CmSetHotSyncExecPath Function

Purpose (*Deprecated*) Sets the path and filename of the HotSync Manager executable for the current Windows user.

Declared In CondMgr.h

Prototype int CmSetHotSyncExecPath (const TCHAR **pPath*)

Parameters → *pPath*

A pointer to a character buffer that contains the path and filename of the HotSync Manager executable on the desktop computer—for example, C:\Program Files\PalmSource\Desktop\HotSync.exe.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_REGISTRY_ACCESS

ERR_UNABLE_TO_SET_CONDUIT_VALUE

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments As of CDK 6.0, this function is deprecated and will be obsoleted in a future version of the Conduit Manager API.

WARNING! Do not use this function. Changing this path could cause legacy conduit installers to fail. Use [CmGetSystemHotSyncExecPath\(\)](#) to retrieve this path, but do not change it.

This function sets the value of the [Core\HotSyncPath](#) configuration entry for the current Windows user. See the description of [CmGetSystemHotSyncExecPath\(\)](#) for details.

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [CmGetHotSyncExecPath\(\)](#),
[CmGetSystemHotSyncExecPath\(\)](#)

Conduit Manager API

CmSetNotifierDll

CmSetNotifierDII Function

Purpose (*Deprecated*) Sets the name of a notifier DLL.

Declared In CondMgr.h

Prototype int CmSetNotifierDII (int *iIndex*,
 const TCHAR **pNotifier*)

Parameters → *iIndex*

The integer index of the notifier whose name you want to establish in the configuration entries.

→ *pNotifier*

The name of the DLL to use for the notifier. Pass NULL to delete the notifier.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_REGISTRY_ACCESS

ERR_UNABLE_TO_SET_CONDUIT_VALUE

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments This function sets the name of a notifier DLL, which is stored as a value in the HotSync Manager portion of the configuration entries. The first notifier configuration entry is named Notifier0, the second entry is named Notifier1, and so on.

IMPORTANT: This function is deprecated. Use one of the Notifier Install Manager functions instead (see [Chapter 13, “Notifier Install Manager API,”](#) on page 909).

CmSetNotifierDII() is retained for backward compatibility, but the Notifier Install Manager enables you more easily to install, uninstall, and modify notifiers.

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [NmRegister\(\)](#), [NmRegisterSystem\(\)](#)

CmSetPCIdentifier Function

Purpose (*Deprecated*) Sets the unique identifier for the current Windows user on the desktop computer.

Declared In CondMgr.h

Prototype int CmSetPCIdentifier (DWORD dwPCID)

Parameters → dwPCID
The DWORD value to use as the identifier for the desktop computer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_REGISTRY_ACCESS

ERR_UNABLE_TO_SET_CONDUIT_VALUE

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments As of CDK 6.0, this function is deprecated and will be obsoleted in a future version of the Conduit Manager API.

WARNING! Do not use this function. Use [CmGetPCIdentifier\(\)](#) to retrieve this value, but do not change it.

This function sets the value of the [HotSync Manager\PCIdentifier](#) configuration entry for the current Windows user. HotSync Manager generates a different PC ID for each Windows user who runs HotSync Manager. Different HotSync users using the same Windows login have the same PC ID.

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [CmGetPCIdentifier\(\)](#)

Conduit Manager API

CmSetSystemBackupConduit

CmSetSystemBackupConduit Function

Purpose	Sets the path or filename of the backup conduit for the system.
Declared In	CondMgr.h
Prototype	<pre>int CmSetSystemBackupConduit (const TCHAR *pConduit)</pre>
Parameters	<p>→ <i>pConduit</i> The full path or filename of the conduit used as the backup conduit.</p>
Returns	<p>If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_REGISTRY_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_POINTER ERR_INSUFFICIENT_PRIVILEGES The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required.</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>
Comments	<p>This function establishes the value of the HotSync Manager\BackupConduit entry used by HotSync Manager in the configuration entries. If you specify a filename that Conduit Manager cannot find (or a NULL filename), this function leaves the filename unchanged and returns no error. Before calling this function, you must check that the file exists where Conduit Manager can find it.</p> <p>This function sets the backup conduit for the system. To set the backup conduit for the current Windows user, call CmSetBackupConduit ().</p>
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	HotSync Manager\BackupConduit , CmGetSystemBackupConduit () , CmSetBackupConduit ()

CmSetSystemCreatorDirectory Function

Purpose Sets the name of the directory in which a system-registered conduit stores its user data.

Declared In CondMgr.h

Prototype

```
int CmSetSystemCreatorDirectory
    (const char *pCreatorID,
     const TCHAR *pDirectory)
```

Parameters

→ *pCreatorID*
The creator ID string of the conduit whose [Directory](#) value you want to set.

→ *pDirectory*
The string value that you want to set for the [Directory](#) configuration entry for the specified conduit.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_AMBIGUOUS_CREATORID

ERR_NO_CONDUIT

ERR_REGISTRY_ACCESS

ERR_UNABLE_TO_SET_CONDUIT_VALUE

ERR_INVALID_CREATOR_ID

ERR_INVALID_POINTER

ERR_INSUFFICIENT_PRIVILEGES

The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required.

ERR_CONDUIT_READ_ONLY

The specified conduit is folder-registered, therefore its registration information is read-only.

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Conduit Manager API

CmSetSystemCreatorDirectory

Comments This function sets the value of the [Directory](#) configuration entry for the specified conduit. The directory string specifies the directory, under the current HotSync user's directory, that the conduit creates to store its user data and support files. For example, if the value of Directory were DateBook and the value of [Core\Path](#) were a typical value, then the full path to this conduit's data directory is:

```
C:\Documents and Settings\<WinUsername>\  
My Documents\Palm OS Desktop\  
<HotSyncUsername>\DateBook
```

This function sets information for a conduit that is registered for the system. To set this information for a conduit that is registered for the current Windows user, call [CmSetCreatorDirectory\(\)](#).

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [Directory](#), [CmGetSystemCreatorDirectory\(\)](#),
[CmSetCreatorDirectory\(\)](#)

CmSetSystemCreatorFile Function

Purpose	Sets the name of the data file for a system-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmSetSystemCreatorFile (const char *pCreatorID, const TCHAR *pFile)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose File value you want to set.</p> <p>→ <i>pFile</i> The string value that you want to set for the File configuration entry for the specified conduit.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER ERR_INSUFFICIENT_PRIVILEGES The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required.</p> <p>ERR_CONDUIT_READ_ONLY The specified conduit is folder-registered, therefore its registration information is read-only.</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Conduit Manager API

CmSetSystemCreatorFile

Comments	This function sets the value of the File configuration entry for the specified conduit. The file string specifies the name of the desktop file to synchronize with the handheld database. This file can be anywhere, but it is commonly located in the directory returned by CmGetCreatorDirectory() . This function sets information for a conduit that is registered for the system. To set this information for a conduit that is registered for the current Windows user, call CmSetCreatorFile() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	File , CmGetSystemCreatorFile() , CmSetCreatorFile()

CmSetSystemCreatorName Function

Purpose	Sets the filename of a system-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmSetSystemCreatorName (const char *pCreatorID, const TCHAR *pConduitName)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Conduit value you want to set.</p> <p>→ <i>pConduitName</i> The string value that you want to set for the Conduit configuration entry for the specified conduit.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER ERR_INSUFFICIENT_PRIVILEGES The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required.</p> <p>ERR_CONDUIT_READ_ONLY The specified conduit is folder-registered, therefore its registration information is read-only.</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Conduit Manager API

CmSetSystemCreatorName

Comments	This function sets the value of the Conduit configuration entry for the specified conduit. This value is the filename of the conduit DLL that is registered with the specified creator ID—for example, conduit filename ToDoCond.dll is registered with the creator ID 'todo'. This function sets information for a conduit that is registered for the system. To set this information for a conduit that is registered for the current Windows user, call CmSetCreatorName() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	Conduit , CmGetSystemCreatorName() , CmSetCreatorName()

CmSetSystemCreatorPriority Function

Purpose	Sets the execution priority for a system-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmSetSystemCreatorPriority (const char *pCreatorID, DWORD dwPriority)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Priority value you want to set.</p> <p>→ <i>dwPriority</i> The value that you want to set for the Priority configuration entry for the specified conduit. This value must be in the range 0 to 4.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER ERR_INSUFFICIENT_PRIVILEGES The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required.</p> <p>ERR_CONDUIT_READ_ONLY The specified conduit is folder-registered, therefore its registration information is read-only.</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Conduit Manager API

CmSetSystemCreatorPriority

Comments	This function retrieves the value of the Priority configuration entry for the specified conduit. HotSync Manager uses the priority value to determine the order in which it executes conduits—that is, the order in which it calls conduits' <code>OpenConduit()</code> entry points. HotSync Manager runs conduits with a value of 0 first and those with 4 last.
	This function sets information for a conduit that is registered for the system. To set this information for a conduit that is registered for the current Windows user, call CmSetCreatorPriority() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	Priority , CmGetSystemCreatorPriority() , CmSetCreatorPriority()

CmSetSystemCreatorRemote Function

Purpose Sets the name of a database on the handheld that is associated with a system-registered conduit.

Declared In CondMgr.h

Prototype

```
int CmSetSystemCreatorRemote
    (const char *pCreatorID,
     const TCHAR *pRemoteDB)
```

Parameters

→ *pCreatorID*
The creator ID string of the conduit whose [Remote](#) value you want to set.

→ *pRemoteDB*
The string value that you want to set for the [Remote](#) configuration entry for the specified conduit.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_AMBIGUOUS_CREATORID

ERR_NO_CONDUIT

ERR_REGISTRY_ACCESS

ERR_UNABLE_TO_SET_CONDUIT_VALUE

ERR_INVALID_CREATOR_ID

ERR_INVALID_POINTER

ERR_INSUFFICIENT_PRIVILEGES

The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required.

ERR_CONDUIT_READ_ONLY

The specified conduit is folder-registered, therefore its registration information is read-only.

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Conduit Manager API

CmSetSystemCreatorRemote

Comments This function sets the value of the [Remote](#) configuration entry for the specified conduit. This value names a single database on the handheld that is associated with this conduit.

NOTE: Though a conduit can access multiple databases on the handheld, `CmSetSystemCreatorRemote()` sets only one handheld database name, which is stored in the `Remote` configuration entry for this conduit. See the description of “[Remote](#)” on page 183 for more information to help you decide whether you want to use this configuration entry.

This function sets information for a conduit that is registered for the system. To set this information for a conduit that is registered for the current Windows user, call [CmSetCreatorRemote\(\)](#).

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [Remote](#), [CmGetSystemCreatorRemote\(\)](#),
[CmSetCreatorRemote\(\)](#)

CmSetSystemCreatorTitle Function

Purpose	Sets the display name of a system-registered conduit.
Declared In	CondMgr.h
Prototype	<pre>int CmSetSystemCreatorTitle (const char *pCreatorID, const TCHAR *pTitle)</pre>
Parameters	<p>→ <i>pCreatorID</i> The creator ID string of the conduit whose Name value you want to set.</p> <p>→ <i>pTitle</i> The string value that you want to set for the Name configuration entry for the specified conduit.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_AMBIGUOUS_CREATORID ERR_NO_CONDUIT ERR_REGISTRY_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_CREATOR_ID ERR_INVALID_POINTER ERR_INSUFFICIENT_PRIVILEGES The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required.</p> <p>ERR_CONDUIT_READ_ONLY The specified conduit is folder-registered, therefore its registration information is read-only.</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Conduit Manager API

CmSetSystemCreatorTitle

Comments	This function sets the value of the Name configuration entry for the specified conduit. When this value is not NULL, HotSync Manager uses it as the conduit's display name in its Custom and HotSync Progress dialog boxes. If this value is NULL, HotSync Manager uses the value returned by a conduit's GetConduitInfo() entry point, and if that fails, its GetConduitName() entry point. This function sets information for a conduit that is registered for the system. To set this information for a conduit that is registered for the current Windows user, call CmSetCreatorTitle() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	Name , CmGetSystemCreatorTitle() , CmSetCreatorTitle()

CmSetSystemCreatorValueDword Function

Purpose Sets the value of a DWORD configuration entry for a system-registered conduit.

Declared In CondMgr.h

Prototype

```
int CmSetSystemCreatorValueDword
    (const char *pCreatorID, TCHAR *pValue,
     DWORD dwValue)
```

Parameters

→ *pCreatorID*

The creator ID string of the conduit whose value you want to set.

→ *pValue*

The configuration entry name.

→ *dwValue*

The value that you want established for the specified configuration entry for the specified conduit.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_AMBIGUOUS_CREATORID

ERR_NO_CONDUIT

ERR_REGISTRY_ACCESS

ERR_INVALID_CREATOR_ID

ERR_INVALID_POINTER

ERR_VALUE_NOT_FOUND

ERR_INSUFFICIENT_PRIVILEGES

The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required.

ERR_CONDUIT_READ_ONLY

The specified conduit is folder-registered, therefore its registration information is read-only.

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Conduit Manager API

CmSetSystemCreatorValueDword

Comments	This function sets a DWORD value in the configuration entries for a conduit. This is a general purpose function for setting a conduit configuration entry value by name. You can use it to set values of configuration entries that you retrieve for your own use with <u>CmGetSystemCreatorValueDword()</u> . This function sets information for a conduit that is registered for the system. To set this information for a conduit that is registered for the current Windows user, call <u>CmSetCreatorValueDword()</u> .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	<u>CmGetSystemCreatorValueDword()</u> , <u>CmSetCreatorValueDword()</u> , <u>CmSetSystemCreatorValueString()</u>

CmSetSystemCreatorValueString Function

Purpose Sets the value of a string configuration entry for a system-registered conduit

Declared In CondMgr.h

Prototype

```
int CmSetSystemCreatorValueString
    (const char *pCreatorID, TCHAR *pValue,
     TCHAR *pString)
```

Parameters

- *pCreatorID*
The creator ID string of the conduit whose value you want to set.
- *pValue*
The configuration entry name.
- *pString*
The string value that you want established for the specified configuration entry for the specified conduit.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_AMBIGUOUS_CREATORID

ERR_NO_CONDUIT

ERR_REGISTRY_ACCESS

ERR_INVALID_CREATOR_ID

ERR_INVALID_POINTER

ERR_VALUE_NOT_FOUND

ERR_INSUFFICIENT_PRIVILEGES

The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required.

ERR_CONDUIT_READ_ONLY

The specified conduit is folder-registered, therefore its registration information is read-only.

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Conduit Manager API

CmSetSystemCreatorValueString

Comments	This function sets a string value in the configuration entries for a conduit. This is a general purpose function for setting a conduit configuration entry value by name. You can use it to set values of configuration entries that you retrieve for your own use with CmGetSystemCreatorValueString() . This function sets information for a conduit that is registered for the system. To set this information for a conduit that is registered for the current Windows user, call CmSetCreatorValueString() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	CmGetSystemCreatorValueString() , CmSetCreatorValueString() , CmGetSystemCreatorValueDword()

FmDisableCurrentUserConduitByIndex **Function**

Purpose	Disables a conduit that is in the current Windows user's Conduits folder, given the conduit's index.
Declared In	CondMgr.h
Prototype	<pre>int FmDisableCurrentUserConduitByIndex (int iIndex)</pre>
Parameters	<p>→ <i>iIndex</i></p> <p>The zero-based index of a folder-based conduit that is registered for the current Windows user. Call FmGetCurrentUserConduitCount() to determine the highest index value.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_FOLDER_NOT_FOUND The specified conduit cannot be disabled because the Conduits or Disabled folder does not exist and cannot be created.</p> <p>ERR_INDEX_OUT_OF_RANGE The specified index value is out of range.</p> <p>For descriptions of all Conduit Manager error codes, see "Conduit Manager Error Codes" on page 843.</p>
Comments	<p>This function moves a folder-registered conduit from the Conduits folder to the Disabled folder for the current Windows user. This action unregisters a conduit so that HotSync Manager cannot call any of its entry points.</p> <p>When this function is called, Conduit Manager checks whether the current Windows user's Conduits and Disabled folders exist before processing the <i>iIndex</i> parameter. If either do not exist, this function returns ERR_FOLDER_NOT_FOUND. Calling FmGetCurrentUserConduitFolder() and FmGetCurrentUserDisabledConduitFolder() creates these folders, if they do not already exist, and returns their paths.</p> <p>If <i>iIndex</i> is within the range of valid index values for folder-based conduits that are currently registered for the current Windows user,</p>

Conduit Manager API

FmDisableCurrentUserConduitByIndex

then Conduit Manager moves the conduit to the Disabled folder. Otherwise, it returns ERR_INDEX_OUT_OF_RANGE.

This function operates only on [folder-registered conduit](#) for the current Windows user. To unregister a conventionally registered conduit, call [CmRemoveConduitByIndex\(\)](#).

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [FmEnableCurrentUserConduitByPath\(\)](#),
[FmGetCurrentUserConduitFolder\(\)](#),
[FmDisableSystemConduitByIndex\(\)](#)

FmDisableCurrentUserConduitByPath Function

Purpose Disables a conduit that is in the current Windows user's Conduits folder, given the conduit's path or filename.

Declared In CondMgr.h

Prototype int FmDisableCurrentUserConduitByPath
(TCHAR **pPath*)

Parameters → *pPath*
A pointer to a character buffer that contains either the full path and filename or only the filename of the conduit in the Conduits folder that you want to disable.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_FOLDER_NOT_FOUND

The specified conduit cannot be disabled because the Conduits or Disabled folder does not exist and cannot be created.

ERR_INVALID_POINTER

For more information about the error codes, see "[Conduit Manager Error Codes](#)" on page 843.

Comments This function moves a folder-registered conduit from the Conduits folder to the Disabled folder for the current Windows user. This action unregisters a conduit so that HotSync Manager cannot call any of its entry points.

When this function is called, Conduit Manager checks whether the current Windows user's Conduits and Disabled folders exist before processing the *pPath* parameter. If either do not exist, this function returns ERR_FOLDER_NOT_FOUND. Calling [FmGetCurrentUserConduitFolder\(\)](#) and [FmGetCurrentUserDisabledConduitFolder\(\)](#) creates these folders, if they do not already exist, and returns their paths.

If *pPath* is only a filename, then Conduit Manager prepends the path of the current Windows user's Conduits folder. Conduit Manager uses the resulting full path and filename to locate the

Conduit Manager API

FmDisableCurrentUserConduitByPath

conduit to disable. If *pPath* is the full path and filename of the conduit to disable, then Conduit Manager assumes *pPath* is the correct path for the current Windows user. If the Conduits folder does not exist, then this function returns
ERR_FOLDER_NOT_FOUND.

This function operates only on [folder-registered conduit](#) for the current Windows user. To unregister a conventionally registered conduit, call [CmRemoveConduitByCreatorID\(\)](#).

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [FmEnableCurrentUserConduitByPath\(\)](#),
[FmGetCurrentUserConduitFolder\(\)](#),
[FmDisableSystemConduitByIndex\(\)](#),
[FmDisableSystemConduitByPath\(\)](#)

FmDisableSystemConduitByIndex Function

Purpose Disables a conduit that is in the system's Conduits folder, given the conduit's index.

Declared In CondMgr.h

Prototype int FmDisableSystemConduitByIndex (int *iIndex*)

Parameters → *iIndex*
The zero-based index of a folder-based conduit that is registered for the system. Call [FmGetSystemConduitCount\(\)](#) to determine the highest index value.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_FOLDER_NOT_FOUND

The specified conduit cannot be disabled because the Conduits or Disabled folder does not exist and cannot be created.

ERR_INDEX_OUT_OF_RANGE

The specified index value is out of range.

ERR_INSUFFICIENT_PRIVILEGES

The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required, but only on an NTFS file system.

For descriptions of all Conduit Manager error codes, see "[Conduit Manager Error Codes](#)" on page 843.

Comments This function moves a folder-registered conduit from the Conduits folder to the Disabled folder for the system. This action unregisters a conduit so that HotSync Manager cannot call any of its entry points.

When this function is called, Conduit Manager checks whether the system's Conduits and Disabled folders exist before processing the *iIndex* parameter. If either do not exist, this function returns ERR_FOLDER_NOT_FOUND. Calling [FmGetSystemConduitFolder\(\)](#) and [FmGetSystemDisabledConduitFolder\(\)](#) creates these folders, if they do not already exist, and returns their paths.

Conduit Manager API

FmDisableSystemConduitByIndex

If *iIndex* is within the range of valid index values for folder-based conduits that are currently registered for the system, then Conduit Manager moves the conduit to the Disabled folder. Otherwise, it returns ERR_INDEX_OUT_OF_RANGE.

This function operates only on [folder-registered conduit](#) for the system. To unregister a conventionally registered conduit, call [CmRemoveSystemConduitByIndex\(\)](#).

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [FmEnableSystemConduitByPath\(\)](#),
[FmGetSystemConduitFolder\(\)](#),
[FmDisableSystemConduitByPath\(\)](#),
[FmDisableCurrentUserConduitByPath\(\)](#)

FmDisableSystemConduitByPath Function

Purpose	Disables a conduit that is in the system's Conduits folder, given the conduit's path or filename.
Declared In	CondMgr.h
Prototype	<code>int FmDisableSystemConduitByPath (TCHAR *pPath)</code>
Parameters	<code>→ pPath</code> A pointer to a character buffer that contains either the full path and filename or only the filename of the conduit in the Conduits folder that you want to disable.
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: <code>ERR_FOLDER_NOT_FOUND</code> The specified conduit cannot be disabled because the Conduits or Disabled folder does not exist and cannot be created. <code>ERR_INVALID_POINTER</code> <code>ERR_INSUFFICIENT_PRIVILEGES</code> The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required, but only on an NTFS file system. For more information about the error codes, see " Conduit Manager Error Codes " on page 843.
Comments	This function moves a folder-registered conduit from the Conduits folder to the Disabled folder for the system. This action unregisters a conduit so that HotSync Manager cannot call any of its entry points. When this function is called, Conduit Manager checks whether the system's Conduits and Disabled folders exist before processing the <i>pPath</i> parameter. If either do not exist, this function returns <code>ERR_FOLDER_NOT_FOUND</code> . Calling FmGetSystemConduitFolder() and FmGetSystemDisabledConduitFolder() creates these folders, if they do not already exist, and returns their paths.

Conduit Manager API

FmDisableSystemConduitByPath

If *pPath* is only a filename, then Conduit Manager prepends the path of the system's Conduits folder. Conduit Manager uses the resulting full path and filename to locate the conduit to disable. If *pPath* is the full path and filename of the conduit to disable, then Conduit Manager assumes *pPath* is the correct path for the system. If the Conduits folder does not exist, then this function returns ERR_FOLDER_NOT_FOUND.

This function operates only on [folder-registered conduit](#) for the system. To unregister a conventionally registered conduit, call [CmRemoveSystemConduitByCreatorID\(\)](#).

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [FmEnableSystemConduitByPath\(\)](#),
[FmGetSystemConduitFolder\(\)](#),
[FmDisableSystemConduitByIndex\(\)](#),
[FmDisableCurrentUserConduitByPath\(\)](#)

FmEnableCurrentUserConduitByPath Function

Purpose	Enables a conduit that is in the current Windows user's Disabled folder, given the conduit's path or filename.
Declared In	CondMgr.h
Prototype	<pre>int FmEnableCurrentUserConduitByPath (TCHAR *pPath)</pre>
Parameters	<p>→ <i>pPath</i></p> <p>A pointer to a character buffer that contains either the full path and filename or only the filename of the conduit in the Disabled folder that you want to enable.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_FOLDER_NOT_FOUND The specified conduit cannot be enabled because the Conduits or Disabled folder does not exist and cannot be created.</p> <p>ERR_INVALID_POINTER</p> <p>For more information about the error codes, see "Conduit Manager Error Codes" on page 843.</p>
Comments	<p>This function moves a folder-registered conduit from the Disabled folder to the Conduits folder for the current Windows user. This action registers a conduit so that HotSync Manager can call any of its entry points.</p> <p>When this function is called, Conduit Manager checks whether the current Windows user's Conduits and Disabled folders exist before processing the <i>pPath</i> parameter. If either do not exist, this function returns ERR_FOLDER_NOT_FOUND. Calling FmGetCurrentUserConduitFolder() and FmGetCurrentUserDisabledConduitFolder() creates these folders, if they do not already exist, and returns their paths.</p> <p>If <i>pPath</i> is only a filename, then Conduit Manager prepends the path of the current Windows user's Disabled folder. Conduit Manager uses the resulting full path and filename to locate the conduit to enable. If <i>pPath</i> is the full path and filename of the conduit to enable, then Conduit Manager assumes <i>pPath</i> is the</p>

Conduit Manager API

FmEnableCurrentUserConduitByPath

correct path for the current Windows user. If the Disabled folder does not exist, then this function returns
ERR_FOLDER_NOT_FOUND.

This function operates only on [folder-registered conduit](#) for the current Windows user. To register a conventionally registered conduit, call [CmInstallConduitByStruct\(\)](#).

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [FmDisableCurrentUserConduitByPath\(\)](#),
[FmGetCurrentUserDisabledConduitFolder\(\)](#),
[FmEnableSystemConduitByPath\(\)](#)

FmEnableSystemConduitByPath Function

Purpose Enables a conduit that is in the system's Disabled folder, given the conduit's path or filename.

Declared In CondMgr.h

Prototype int FmEnableSystemConduitByPath (TCHAR **pPath*)

Parameters → *pPath*

A pointer to a character buffer that contains either the full path and filename or only the filename of the conduit in the Disabled folder that you want to enable.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_FOLDER_NOT_FOUND

The specified conduit cannot be enabled because the Conduits or Disabled folder does not exist and cannot be created.

ERR_INVALID_POINTER

ERR_INSUFFICIENT_PRIVILEGES

The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required, but only on an NTFS file system.

For more information about the error codes, see "[Conduit Manager Error Codes](#)" on page 843.

Comments This function moves a folder-registered conduit from the Disabled folder to the Conduits folder for the system. This action registers a conduit so that HotSync Manager can call any of its entry points.

When this function is called, Conduit Manager checks whether the system's Conduits and Disabled folders exist before processing the *pPath* parameter. If either do not exist, this function returns ERR_FOLDER_NOT_FOUND. Calling

[FmGetSystemConduitFolder\(\)](#) and

[FmGetSystemDisabledConduitFolder\(\)](#) creates these folders, if they do not already exist, and returns their paths.

If *pPath* is only a filename, then Conduit Manager prepends the path of the system's Disabled folder. Conduit Manager uses the

Conduit Manager API

FmEnableSystemConduitByPath

resulting full path and filename to locate the conduit to enable. If *pPath* is the full path and filename of the conduit to enable, then Conduit Manager assumes *pPath* is the correct path for the system. If the Disabled folder does not exist, then this function returns ERR_FOLDER_NOT_FOUND.

This function operates only on [folder-registered conduit](#) for the system. To register a conventionally registered conduit, call [CmInstallSystemConduitByStruct\(\)](#).

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [FmDisableSystemConduitByPath\(\)](#),
[FmGetSystemDisabledConduitFolder\(\)](#),
[FmEnableCurrentUserConduitByPath\(\)](#)

FmGetCurrentUserConduitByIndex Function

Purpose Retrieves information about a folder-based conduit registered for the current Windows user, given the conduit's index.

Declared In CondMgr.h

Prototype

```
int FmGetCurrentUserConduitByIndex (int iIndex,  
                                    CmConduitType2 &sConduitInfo)
```

Parameters

→ *iIndex*
The zero-based index of a folder-based conduit that is registered for the current Windows user. Call [FmGetCurrentUserConduitCount\(\)](#) to determine the highest index value.

← *sConduitInfo*
A reference to a [CmConduitType2](#) structure that describes the specified conduit.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_INDEX_OUT_OF_RANGE
The specified index value is out of range.

ERR_FOLDER_NOT_FOUND
The specified conduit cannot be enabled because the Conduits or Disabled folder does not exist and cannot be created.

For descriptions of all Conduit Manager error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments This function retrieves information about a folder-based conduit that is registered for the current Windows user. To retrieve this information for a folder-based conduit that is registered for the system, call [FmGetSystemConduitByIndex\(\)](#).

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [CmConduitType2](#), [FmGetCurrentUserConduitCount\(\)](#), [FmGetSystemConduitByIndex\(\)](#)

Conduit Manager API

FmGetCurrentUserConduitCount

FmGetCurrentUserConduitCount Function

Purpose	Returns the number of folder-based conduits that are registered with HotSync Manager for the current Windows user.
Declared In	CondMgr.h
Prototype	<code>int FmGetCurrentUserConduitCount (void)</code>
Parameters	None.
Returns	If successful, returns the number of registered conduits. If unsuccessful, returns the following nonzero error code value: <code>ERR_FOLDER_NOT_FOUND</code> The specified conduit cannot be enabled because the Conduits or Disabled folder does not exist and cannot be created. For descriptions of all Conduit Manager error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	The returned count includes <i>only</i> folder-based conduits that are registered for the current Windows user. This count does not include system-registered conduits. To retrieve this information for conduits that are registered for the system, call FmGetSystemConduitCount () .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	FmGetSystemConduitCount () , FmGetCurrentUserConduitByIndex () , FmDisableCurrentUserConduitByIndex ()

FmGetCurrentUserConduitFolder Function

Purpose	Retrieves the path of the current Windows user's Conduits folder.
Declared In	CondMgr.h
Prototype	<pre>int FmGetCurrentUserConduitFolder (TCHAR *pPath, int *piSize)</pre>
Parameters	<p>$\leftarrow pPath$ A pointer to a character buffer. Upon return, this contains the requested path.</p> <p>$\leftrightarrow piSize$ A pointer to an integer value that specifies the size of the character buffer referenced by $pPath$. Upon return, this is the actual size, in TCHARs, of the path string. If this function fails because the string buffer is too small, the value of $piSize$ upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_FOLDER_NOT_FOUND The Conduits folder does not exist and cannot be created.</p> <p>ERR_INVALID_POINTER One or more input parameters are NULL or are otherwise invalid pointers.</p> <p>ERR_BUFFER_TOO_SMALL The supplied buffer is too small. This function passes back the required size via the $piSize$ parameter upon return.</p> <p>For more information about the error codes, see "Conduit Manager Error Codes" on page 843.</p>

Conduit Manager API

FmGetCurrentUserConduitFolder

Comments	This function checks whether the Conduits folder exists for the current Windows user. If the folder exists, then Conduit Manager retrieves the path. If the folder does not exist, Conduit Manager attempts to create a Conduits folder for the current Windows user. Therefore, if this function passes back a path, then the Conduits folder exists.
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	FmGetCurrentUserDisabledConduitFolder() , FmGetSystemConduitFolder()

FmGetCurrentUserDisabledConduitFolder Function

Purpose Retrieves the path of the current Windows user's Disabled folder.

Declared In CondMgr.h

Prototype `int FmGetCurrentUserDisabledConduitFolder
 (TCHAR *pPath, int *piSize)`

Parameters $\leftarrow pPath$
A pointer to a character buffer. Upon return, this contains the requested path.

$\leftrightarrow piSize$
A pointer to an integer value that specifies the size of the character buffer referenced by *pPath*. Upon return, this is the actual size, in TCHARs, of the path string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

`ERR_FOLDER_NOT_FOUND`

The Disabled folder does not exist and cannot be created.

`ERR_INVALID_POINTER`

One or more input parameters are NULL or are otherwise invalid pointers.

`ERR_BUFFER_TOO_SMALL`

The supplied buffer is too small. This function passes back the required size via the *piSize* parameter upon return.

For more information about the error codes, see "[Conduit Manager Error Codes](#)" on page 843.

Conduit Manager API

FmGetCurrentUserDisabledConduitFolder

Comments	This function checks whether the Disabled folder exists for the current Windows user. If the folder exists, then Conduit Manager retrieves the path. If the folder does not exist, Conduit Manager attempts to create a Disabled folder for the current Windows user. Therefore, if this function passes back a path, then the Disabled folder exists.
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	FmGetCurrentUserConduitFolder() , FmGetSystemDisabledConduitFolder()

FmGetSystemConduitByIndex Function

Purpose	Retrieves information about a folder-based conduit registered for the system, given the conduit's index.
Declared In	CondMgr.h
Prototype	<pre>int FmGetSystemConduitByIndex (int iIndex, CmConduitType2 &sConduitInfo)</pre>
Parameters	<p>→ <i>iIndex</i> The zero-based index of a folder-based conduit that is registered for the system. Call FmGetSystemConduitCount() to determine the highest index value.</p> <p>← <i>sConduitInfo</i> A reference to a CmConduitType2 structure that describes the specified conduit.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_INDEX_OUT_OF_RANGE The specified index value is out of range.</p> <p>ERR_FOLDER_NOT_FOUND The specified conduit cannot be enabled because the Conduits or Disabled folder does not exist and cannot be created.</p> <p>For descriptions of all Conduit Manager error codes, see “Conduit Manager Error Codes” on page 843.</p>
Comments	This function retrieves information about a folder-based conduit that is registered for the system. To retrieve this information for a folder-based conduit that is registered for the current Windows user, call FmGetCurrentUserConduitByIndex() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	CmConduitType2 , FmGetSystemConduitCount() , FmGetCurrentUserConduitByIndex()

Conduit Manager API

FmGetSystemConduitCount

FmGetSystemConduitCount Function

Purpose	Returns the number of folder-based conduits that are registered with HotSync Manager for the system.
Declared In	CondMgr.h
Prototype	<code>int FmGetSystemConduitCount (void)</code>
Parameters	None.
Returns	If successful, returns the number of registered conduits. If unsuccessful, returns the following nonzero error code value: <code>ERR_FOLDER_NOT_FOUND</code> The specified conduit cannot be enabled because the Conduits or Disabled folder does not exist and cannot be created. For descriptions of all Conduit Manager error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	The returned count includes <i>only</i> folder-based conduits that are registered for the system. This count does not include user-registered conduits. To retrieve this information for conduits that are registered for the current Windows user, call FmGetCurrentUserConduitCount () .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	FmGetCurrentUserConduitCount () , FmGetSystemConduitByIndex () , FmDisableSystemConduitByIndex ()

FmGetSystemConduitFolder Function

Purpose	Retrieves the path of the system's Conduits folder.
Declared In	CondMgr.h
Prototype	<pre>int FmGetSystemConduitFolder (TCHAR *pPath, int *piSize)</pre>
Parameters	<p>$\leftarrow pPath$ A pointer to a character buffer. Upon return, this contains the requested path.</p> <p>$\leftrightarrow piSize$ A pointer to an integer value that specifies the size of the character buffer referenced by <i>pPath</i>. Upon return, this is the actual size, in TCHARs, of the path string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_FOLDER_NOT_FOUND The Conduits folder does not exist and cannot be created.</p> <p>ERR_INVALID_POINTER One or more input parameters are NULL or are otherwise invalid pointers.</p> <p>ERR_BUFFER_TOO_SMALL The supplied buffer is too small. This function passes back the required size via the <i>piSize</i> parameter upon return.</p> <p>For more information about the error codes, see "Conduit Manager Error Codes" on page 843.</p>

Conduit Manager API

FmGetSystemConduitFolder

Comments	This function checks whether the Conduits folder exists for the system. If the folder exists, then Conduit Manager retrieves the path. If the folder does not exist, Conduit Manager attempts to create a Conduits folder for the system. Therefore, if this function passes back a path, then the Conduits folder exists.
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	<u>FmGetSystemDisabledConduitFolder()</u> , <u>FmGetCurrentUserConduitFolder()</u>

FmGetSystemDisabledConduitFolder Function

Purpose	Retrieves the path of the system's Disabled folder.
Declared In	CondMgr.h
Prototype	<pre>int FmGetSystemDisabledConduitFolder (TCHAR *pPath, int *piSize)</pre>
Parameters	<p>$\leftarrow pPath$ A pointer to a character buffer. Upon return, this contains the requested path.</p> <p>$\leftrightarrow piSize$ A pointer to an integer value that specifies the size of the character buffer referenced by $pPath$. Upon return, this is the actual size, in TCHARs, of the path string. If this function fails because the string buffer is too small, the value of $piSize$ upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_FOLDER_NOT_FOUND The Disabled folder does not exist and cannot be created.</p> <p>ERR_INVALID_POINTER One or more input parameters are NULL or are otherwise invalid pointers.</p> <p>ERR_BUFFER_TOO_SMALL The supplied buffer is too small. This function passes back the required size via the $piSize$ parameter upon return.</p> <p>For more information about the error codes, see "Conduit Manager Error Codes" on page 843.</p>

Conduit Manager API

FmGetSystemDisabledConduitFolder

Comments	This function checks whether the Disabled folder exists for the system. If the folder exists, then Conduit Manager retrieves the path. If the folder does not exist, Conduit Manager attempts to create a Disabled folder for the system. Therefore, if this function passes back a path, then the Disabled folder exists.
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	<u>FmGetSystemConduitFolder()</u> , <u>FmGetCurrentUserDisabledConduitFolder()</u>

Conduit Manager Error Codes

[Table 11.1](#) lists the values of error codes that Conduit Manager functions can return. The description of each function states which errors each function can return.

All of the named error codes below are defined as preprocessor constants, which are declared in the CondMgr.h header file.

Table 11.1 Conduit Manager error codes

Value	Error Code	Description
-1029	ERR_CONFLICTING_EXTENSION	The specified filename extension is already registered for another install conduit.
-1028	ERR_COULD_NOT_CREATE_DIRECTORY	The Core\Path configuration entry could not be set to the specified value because the directory could not be created.
-1027	ERR_AMBIGUOUS_CREATORID	One or more folder-based conduits are already registered with the specified creator ID. Resolve the conflict and call this function again. See “ Resolving Conduit Conflicts ” on page 117 in the <i>C/C++ Sync Suite Companion</i> .
-1026	ERR_CONDUIT_READ_ONLY	The specified conduit is folder-registered, therefore its registration information is read-only. All CmSetCreator...() and CmSetSystemCreator...() functions can return this error.
-1025	ERR_FOLDER_NOT_FOUND	The Conduits or Disabled folder does not exist and cannot be created.

Conduit Manager API

Conduit Manager Error Codes

Table 11.1 Conduit Manager error codes (*continued*)

Value	Error Code	Description
-1024	ERR_INSUFFICIENT_PRIVILEGES	The current Windows user does not have sufficient privileges to perform this operation. Administrator privileges are required to make system-level changes but not to read system-level information. This error can be returned by a call that moves a conduit in the system-level area, but only if the user does not have administrator privileges and has an NTFS file system.
-1023	ERR_INVALID_INSTALL_ID	The specified unique ID for an install conduit is not valid.
-1022	ERR_INSTALL_ID_ALREADY_IN_USE	The specified unique ID for an install conduit is already in use.
-1021	ERR_NOTIFIER_NOT_FOUND	The specified notifier is not registered.
-1019	ERR_ALREADY_INSTALLED	The specified conduit or notifier is already installed.
-1018	ERR_INVALID_PATH	The specified path is not valid.
-1017	ERR_NO_LONGER_SUPPORTED	The value or feature that you specified is no longer supported.
-1016	ERR_INVALID_COM_PORT_TYPE	The specified communications port type is not valid. You must use one of the values described in “ Communications Ports ” on page 662.

Table 11.1 Conduit Manager error codes (*continued*)

Value	Error Code	Description
-1015	ERR_INVALID_CONDUIT_TYPE	The specified conduit type is not valid. You must use one of the values described in “ Conduit Information Types ” on page 661.
-1014	ERR_UNABLE_TO_INSTALL_OLD	No function returns this error code.
-1013	ERR_INVALID_POINTER	The specified pointer is not a valid pointer.
-1012	ERR_INVALID_CREATOR_ID	The specified conduit creator ID is not valid.
-1011	ERR_VALUE_NOT_FOUND	The specified value could not be found in the configuration entries for this conduit.
-1010	ERR_BUFFER_TOO_SMALL	The buffer is too small to hold the requested information.
-1009	ERR_INVALID_HANDLE	The specified structure handle is not valid.
-1008	ERR_UNABLE_TO_SET_CONDUIT_VALUE	The specified conduit configuration entry could not be set.
-1007	ERR_UNABLE_TO_CREATE_CONDUIT	The conduit could not be registered with HotSync Manager.
-1006	ERR_REGISTRY_ACCESS	Unable to access the conduit configuration entries.
-1006	ERR_STORAGE_ACCESS	Unable to access the conduit configuration entries.

Conduit Manager API

Conduit Manager Error Codes

Table 11.1 Conduit Manager error codes (*continued*)

Value	Error Code	Description
-1005	ERR_CREATORID_ALREADY_IN_USE	The creator ID that you specified to use as a new creator ID in the configuration entry is already in use.
-1004	ERR_NO_MEMORY	Not enough memory is available to perform the requested operation.
-1003	ERR_NO_CONDUIT	The specified conduit does not exist.
-1002	ERR_UNABLE_TO_DELETE	The configuration entries for the specified conduit could not be deleted.
-1001	ERR_INDEX_OUT_OF_RANGE	The specified index value is out of range.
-1000	ERR_CONDUIT_MGR	An nonspecific Conduit Manager error occurred.

12

Install Conduit Manager API

The Install Conduit Manager provides an API for registering an install conduit with HotSync® Manager. An **install conduit** is one that can install databases in the handheld's main memory or files on an expansion card.

This API also provides utility functions for accessing the install conduit configuration entries that the Install Conduit Manager creates when you register an install conduit. For a definition of each configuration entry, see [Appendix A, “Configuration Entries,”](#) on page 175 in the *Introduction to Conduit Development*.

The Install Conduit Manager functions are available in CondMgr .d11 (which also includes the Conduit Manager functions) and are declared in CondMgr .h.

The sections in this chapter are:

Install Conduit Manager Structures	848
Install Conduit Manager Functions	850

For more information on the Install Conduit Manager, see “[Using the Install Conduit Manager API](#)” on page 158 in the *C/C++ Sync Suite Companion*.

Install Conduit Manager API

Install Conduit Manager Structures

Install Conduit Manager Structures

This section describes the [FileInstallType](#) data structure that you use with the [ImRegister\(\)](#) and [ImRegisterSystem\(\)](#) functions.

FileInstallType Struct

Purpose	Specifies registration information about your install conduit when you call ImRegister() or ImRegisterSystem() to register it.
Declared In	CondMgr.h
Prototype	<pre>typedef struct { TCHAR szDir[64]; TCHAR szExt[256]; DWORD dwMask; TCHAR szModule[256]; DWORD dwCreatorID; TCHAR szName[256]; } FileInstallType;</pre>
Fields	<p>szDir Specifies the name of the handheld-install directory associated with this install conduit—for example, “Install”. This value is set in an install conduit’s Directory configuration entry.</p> <p>szExt Specifies the file type extensions supported by this install conduit. This value is set in an install conduit’s Extensions configuration entry.</p> <p>dwMask Specifies a unique bit mask associated with this install conduit. This value is set in an install conduit’s Mask configuration entry.</p> <p>szModule Specifies the filename of this install conduit DLL—for example, “inscn20.dll”. This value is set in an install conduit’s Module configuration entry.</p>

dwCreatorID

Specifies the unique ID associated with this install conduit. This value is set in an install conduit's [CreatorID](#) configuration entry.

szName

Specifies the display name of this install conduit—for example, "Install Conduit". This value is set in an install conduit's [Name](#) configuration entry.

Comments Each of these fields corresponds to an install conduit configuration entry described in [Table A.2](#) on page 185.

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [ImRegister\(\)](#), [ImRegisterSystem\(\)](#)

Install Conduit Manager API

Install Conduit Manager Functions

Install Conduit Manager Functions

This section describes the following functions, which allow you to register install conduits and to retrieve information from the install conduit [configuration entries](#).

<code>ImGetDirectory()</code>	Retrieves the name of the directory in which a user-registered install conduit looks for files to install.
<code>ImGetDWord()</code>	Retrieves a DWORD configuration entry value for a user-registered install conduit.
<code>ImGetExtension()</code>	Retrieves the file extensions supported by a user-registered install conduit.
<code>ImGetMask()</code>	Retrieves the value of the bit mask for a user-registered install conduit.
<code>ImGetModule()</code>	Retrieves the filename of a user-registered install conduit.
<code>ImGetName()</code>	Retrieves the display name of a user-registered install conduit.
<code>ImGetString()</code>	Retrieves a string value configuration entry for a user-registered install conduit.
<code>ImGetSystemDirectory()</code>	Retrieves the name of the directory in which a system-registered install conduit looks for files to install.
<code>ImGetSystemDWord()</code>	Retrieves a DWORD configuration entry value for a system-registered install conduit.
<code>ImGetSystemExtension()</code>	Retrieves the file extensions supported by a system-registered install conduit.
<code>ImGetSystemMask()</code>	Retrieves the value of the bit mask for a system-registered install conduit.
<code>ImGetSystemModule()</code>	Retrieves the filename of a system-registered install conduit.
<code>ImGetSystemName()</code>	Retrieves the display name of a system-registered install conduit.

<u>ImGetSystemString()</u>	Retrieves a string value configuration entry for a system-registered install conduit.
<u>ImRegister()</u>	Registers an install conduit with HotSync Manager for the current Windows user. Uses a <code>FileInstallType</code> structure.
<u>ImRegisterID()</u>	Registers a new unique ID for an install conduit with HotSync Manager for the current Windows user (requires calling other <code>ImSet...</code> () functions to complete registration).
<u>ImRegisterSystem()</u>	Registers an install conduit with HotSync Manager for the system. Uses a <code>FileInstallType</code> structure.
<u>ImRegisterSystemID()</u>	Registers a new unique ID for an install conduit with HotSync Manager for the system (requires calling other <code>ImSetSystem...</code> () functions to complete registration).
<u>ImSetDirectory()</u>	Sets the name of the directory in which a user-registered install conduit looks for files to install.
<u>ImSetDWord()</u>	Sets a DWORD configuration entry value for a user-registered install conduit.
<u>ImSetExtension()</u>	Sets the file extensions supported by a user-registered install conduit.
<u>ImSetMask()</u>	Sets the value of the bit mask for a user-registered install conduit.
<u>ImSetModule()</u>	Sets the filename of a user-registered install conduit.
<u>ImSetName()</u>	Sets the display name of a user-registered install conduit.
<u>ImSetString()</u>	Sets a string value configuration entry for a user-registered install conduit.
<u>ImSetSystemDirectory()</u>	Sets the name of the directory in which a system-registered install conduit looks for files to install.
<u>ImSetSystemDWord()</u>	Sets a DWORD configuration entry value for a system-registered install conduit.

Install Conduit Manager API

Install Conduit Manager Functions

<u>ImSetSystemExtension()</u>	Sets the file extensions supported by a system-registered install conduit.
<u>ImSetSystemMask()</u>	Sets the value of the bit mask for a system-registered install conduit.
<u>ImSetSystemModule()</u>	Sets the filename of a system-registered install conduit.
<u>ImSetSystemName()</u>	Sets the display name of a system-registered install conduit.
<u>ImSetSystemString()</u>	Sets a string value configuration entry for a system-registered install conduit.
<u>ImUnregisterID()</u>	Unregisters an install conduit that is registered for the current Windows user.
<u>ImUnregisterSystemID()</u>	Unregisters an install conduit that is registered for the system.

ImGetDirectory Function

Purpose Retrieves the name of the directory in which a user-registered install conduit looks for files to install.

Declared In CondMgr.h

Prototype

```
int WINAPI ImGetDirectory (DWORD dwID,
                           TCHAR *pDirectory, int *piSize)
```

Parameters

→ *dwID*
A DWORD that specifies the ID of the install conduit whose Directory value you want to retrieve.

← *pDirectory*
A pointer to a null-terminated character buffer. Upon return, this is the value of the [Directory](#) configuration entry for the specified install conduit.

↔ *piSize*
A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the *pDirectory* parameter. Upon return, this is the actual size, in TCHARS, of the Directory string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_NO_CONDUIT

ERR_STORAGE_ACCESS

ERR_BUFFER_TOO_SMALL

The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the *piSize* parameter.

ERR_VALUE_NOT_FOUND

ERR_INVALID_INSTALL_ID

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Install Conduit Manager API

ImGetDirectory

Comments This function retrieves the value of the [Directory](#) configuration entry for the specified install conduit. The directory string specifies the directory, under the current HotSync user's directory, in which an install conduit looks for files to install. For example, if the value of Directory were Install and the value of [Core\Path](#) were a typical value, then the full path to this install conduit's directory is:

```
C:\Documents and Settings\<WinUsername>\  
My Documents\Palm OS Desktop\  
<HotSyncUsername>\Install
```

This function retrieves information about an install conduit that is registered for the current Windows user. To retrieve this information for an install conduit that is registered for the system, call [ImGetSystemDirectory\(\)](#).

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [Directory](#), [ImSetDirectory\(\)](#)

ImGetDWord Function

Purpose	Retrieves a DWORD configuration entry value for a user-registered install conduit.
Declared In	<code>CondMgr.h</code>
Prototype	<pre>int WINAPI ImGetDWord (DWORD dwID, const TCHAR *pValue, DWORD *pdwValue, DWORD dwDefault)</pre>
Parameters	<p>→ <i>dwID</i> A DWORD that specifies the ID of the install conduit whose configuration entry value you want to retrieve.</p> <p>→ <i>pValue</i> A pointer to a null-terminated character buffer containing the string name of the configuration entry.</p> <p>← <i>pdwValue</i> A pointer to a DWORD. Upon return, this contains the value of the specified configuration entry.</p> <p>→ <i>dwDefault</i> The default value to return in <i>*pdwValue</i> if the specified <i>pValue</i> name could not be found in the configuration entries.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_NO_CONDUIT ERR_STORAGE_ACCESS ERR_VALUE_NOT_FOUND ERR_INVALID_INSTALL_ID ERR_INVALID_POINTER</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Install Conduit Manager API

ImGetDWord

Comments	This function retrieves a DWORD value from the configuration entries for an install conduit. This is a general purpose function for retrieving the value of an install conduit configuration entry by name. You can use it to retrieve values of configuration entries that you create for your own use with ImSetDWord() . If Install Conduit Manager does not find the configuration entry name specified by <i>pValue</i> , it sets <i>*pdwValue</i> equal to the value of <i>dwDefault</i> and returns 0 (success). However, if the install conduit specified by <i>dwID</i> does not exist, this function does <i>not</i> set <i>*pdwValue</i> equal to the value of <i>dwDefault</i> but does return an error. This function retrieves information about an install conduit that is registered for the current Windows user. To retrieve this information for an install conduit that is registered for the system, call ImGetSystemDWord() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	ImSetDWord()

ImGetExtension Function

Purpose	Retrieves the file extensions supported by a user-registered install conduit.
Declared In	<code>CondMgr.h</code>
Prototype	<pre>int WINAPI ImGetExtension (DWORD dwID, TCHAR *pExtension, int *piSize)</pre>
Parameters	<p>→ <i>dwID</i> A DWORD that specifies the ID of the install conduit whose Extension value you want to retrieve.</p> <p>← <i>pExtension</i> A pointer to a null-terminated character buffer. Upon return, this is the value of the Extensions configuration entry for the specified install conduit.</p> <p>↔ <i>piSize</i> A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the <i>pExtension</i> parameter. Upon return, this is the actual size, in TCHARS, of the Extension string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_NO_CONDUIT ERR_STORAGE_ACCESS ERR_BUFFER_TOO_SMALL The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the <i>piSize</i> parameter.</p> <p>ERR_VALUE_NOT_FOUND ERR_INVALID_INSTALL_ID ERR_INVALID_POINTER</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Install Conduit Manager API

ImGetExtension

Comments	This function retrieves the value of the Extensions configuration entry for the specified install conduit. The Extensions string specifies the file extensions that the install conduit supports.
	This function retrieves information about an install conduit that is registered for the current Windows user. To retrieve this information for an install conduit that is registered for the system, call ImGetSystemExtension() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	Extensions , ImSetExtension()

ImGetMask Function

Purpose Retrieves the value of the bit mask for a user-registered install conduit.

Declared In CondMgr.h

Prototype int WINAPI ImGetMask (DWORD *dwID*, DWORD **pdwMask*)

Parameters
→ *dwID*
A DWORD that specifies the ID of the install conduit whose Mask value you want to retrieve.

← *pdwMask*
A pointer to a DWORD. Upon return, this is the value of the [Mask](#) configuration entry for the specified install conduit.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_NO_CONDUIT
ERR_STORAGE_ACCESS
ERR_VALUE_NOT_FOUND
ERR_INVALID_INSTALL_ID
ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments This function retrieves the value of the [Mask](#) configuration entry for the specified install conduit. The Mask is a bit mask value that uniquely identifies an install conduit.

This function retrieves information about an install conduit that is registered for the current Windows user. To retrieve this information for an install conduit that is registered for the system, call [ImGetSystemMask\(\)](#).

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [Mask](#), [ImSetMask\(\)](#), “[Registering an Install Conduit](#)” on page 159 in the *C/C++ Sync Suite Companion*

Install Conduit Manager API

ImGetModule

ImGetModule Function

Purpose Retrieves the filename of a user-registered install conduit.

Declared In CondMgr.h

Prototype `int WINAPI ImGetModule (DWORD dwID,
TCHAR *pModule, int *piSize)`

Parameters

→ *dwID*
A DWORD that specifies the ID of the install conduit whose Module value you want to retrieve.

← *pModule*
A pointer to a null-terminated character buffer. Upon return, this is the value of the [Module](#) configuration entry for the specified conduit.

↔ *piSize*
The size, in TCHARS, of the buffer referenced by the *pModule* parameter. Upon return, this is the actual size, in TCHARS, of the Module string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

`ERR_NO_CONDUIT`

`ERR_STORAGE_ACCESS`

`ERR_BUFFER_TOO_SMALL`

The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the *piSize* parameter.

`ERR_INVALID_INSTALL_ID`

`ERR_INVALID_POINTER`

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments	This function retrieves the value of the Module configuration entry for the specified install conduit. The Module string is the filename of the install conduit DLL to load when the install conduit is invoked. This function retrieves information about an install conduit that is registered for the current Windows user. To retrieve this information for an install conduit that is registered for the system, call _ImGetSystemModule() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	Module , _ImSetModule()

Install Conduit Manager API

ImGetName

ImGetName Function

Purpose Retrieves the display name of a user-registered install conduit.

Declared In CondMgr.h

Prototype `int WINAPI ImGetName (DWORD dwID,
 TCHAR *pConduitName, int *piSize)`

Parameters → *dwID*
A DWORD that specifies the ID of the install conduit whose Name value you want to retrieve.

← *pConduitName*
A pointer to a null-terminated character buffer. Upon return, this is the value of the [Name](#) configuration entry for the specified install conduit.

↔ *piSize*
A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the *pConduitName* parameter.
Upon return, this is the actual size, in TCHARS, of the Name string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

`ERR_NO_CONDUIT`

`ERR_STORAGE_ACCESS`

`ERR_BUFFER_TOO_SMALL`

The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the *piSize* parameter.

`ERR_INVALID_INSTALL_ID`

`ERR_INVALID_POINTER`

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments This function retrieves the value of the [Name](#) configuration entry for the specified install conduit. The Name string specifies the name of the install conduit to display in HotSync Manager's **Custom** dialog box.

This function retrieves information about an install conduit that is registered for the current Windows user. To retrieve this information for an install conduit that is registered for the system, call [_ImGetSystemName \(\)](#).

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [Name](#), [_ImSetName \(\)](#)

ImGetString Function

Purpose	Retrieves a string value configuration entry for a user-registered install conduit.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI ImGetString (DWORD dwID, const TCHAR *pValue, TCHAR *pString, int *piSize, const TCHAR *pDefault)</pre>
Parameters	<p>→ <i>dwID</i> A DWORD that specifies the ID of the install conduit for which you want to retrieve a configuration entry value.</p> <p>→ <i>pValue</i> The configuration entry name.</p> <p>← <i>pString</i> A pointer to a character buffer. Upon return, this is the value of the specified configuration entry for the specified conduit.</p> <p>↔ <i>piSize</i> A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the <i>pString</i> parameter. Upon return, this is the actual size, in TCHARS, of the string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p> <p>→ <i>pDefault</i> The default value to return in <i>*pString</i> if the specified <i>pValue</i> name could not be found in the configuration entries.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_BUFFER_TOO_SMALL The size of the buffer that you supplied is too small to contain the string, so this function passes back the actual required size into the <i>piSize</i> parameter.</p> <p>ERR_NO_CONDUIT</p> <p>ERR_STORAGE_ACCESS</p> <p>ERR_VALUE_NOT_FOUND</p>

ERR_INVALID_INSTALL_ID

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments

This function retrieves a string value from the configuration entries for an install conduit. This is a general purpose function for retrieving an install conduit configuration entry value by name. You can use it to retrieve values of configuration entries that you create for your own use with [ImSetString\(\)](#).

If Install Conduit Manager does not find the configuration entry name specified by *pValue*, it passes back the default value (sets **pString* equal to the value of **pDefault*) and returns 0 (success). However, if the install conduit specified by *dwID* does not exist, Install Conduit Manager does *not* pass back the default value but does return an error.

This function retrieves information about an install conduit that is registered for the current Windows user. To retrieve this information for an install conduit that is registered for the system, call [ImGetSystemString\(\)](#).

Compatibility

Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also

[ImSetString\(\)](#)

Install Conduit Manager API

ImGetSystemDirectory

ImGetSystemDirectory Function

Purpose	Retrieves the name of the directory in which a system-registered install conduit looks for files to install.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI ImGetSystemDirectory (DWORD dwID, TCHAR *pDirectory, int *piSize)</pre>
Parameters	<p>→ <i>dwID</i> A DWORD that specifies the ID of the install conduit whose Directory value you want to retrieve.</p> <p>← <i>pDirectory</i> A pointer to a null-terminated character buffer. Upon return, this is the value of the Directory configuration entry for the specified install conduit.</p> <p>↔ <i>piSize</i> A pointer to an integer that specifies the size, in TCHARs, of the buffer referenced by the <i>pDirectory</i> parameter. Upon return, this is the actual size, in TCHARs, of the Directory string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_NO_CONDUIT ERR_STORAGE_ACCESS ERR_BUFFER_TOO_SMALL The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the <i>piSize</i> parameter.</p> <p>ERR_VALUE_NOT_FOUND ERR_INVALID_INSTALL_ID ERR_INVALID_POINTER</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments This function retrieves the value of the [Directory](#) configuration entry for the specified install conduit. The directory string specifies the directory, under the current HotSync user's directory, in which an install conduit looks for files to install. For example, if the value of Directory were Install and the value of [Core\Path](#) were a typical value, then the full path to this install conduit's directory is:

```
C:\Documents and Settings\<WinUsername>\  
My Documents\Palm OS Desktop\  
<HotSyncUsername>\Install
```

This function retrieves information about an install conduit that is registered for the system. To retrieve this information for an install conduit that is registered for the current Windows user, call [ImGetDirectory\(\)](#).

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [Directory](#), [ImSetSystemDirectory\(\)](#)

Install Conduit Manager API

ImGetSystemDWord

ImGetSystemDWord Function

Purpose	Retrieves a DWORD configuration entry value for a system-registered install conduit.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI ImGetSystemDWord (DWORD dwID, const TCHAR *pValue, DWORD *pdwValue, DWORD dwDefault)</pre>
Parameters	<p>→ <i>dwID</i> A DWORD that specifies the ID of the install conduit whose configuration entry value you want to retrieve.</p> <p>→ <i>pValue</i> A pointer to a null-terminated character buffer containing the string name of the configuration entry.</p> <p>← <i>pdwValue</i> A pointer to a DWORD. Upon return, this contains the value of the specified configuration entry.</p> <p>→ <i>dwDefault</i> The default value to return in <i>*pdwValue</i> if the specified <i>pValue</i> name could not be found in the configuration entries.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_NO_CONDUIT ERR_STORAGE_ACCESS ERR_VALUE_NOT_FOUND ERR_INVALID_INSTALL_ID ERR_INVALID_POINTER</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments	This function retrieves a DWORD value from the configuration entries for an install conduit. This is a general purpose function for retrieving the value of an install conduit configuration entry by name. You can use it to retrieve values of configuration entries that you create for your own use with ImSetDWord() . If Install Conduit Manager does not find the configuration entry name specified by <i>pValue</i> , it sets <i>*pdwValue</i> equal to the value of <i>dwDefault</i> and returns 0 (success). However, if the install conduit specified by <i>dwID</i> does not exist, this function does <i>not</i> set <i>*pdwValue</i> equal to the value of <i>dwDefault</i> but does return an error. This function retrieves information about an install conduit that is registered for the system. To retrieve this information for an install conduit that is registered for the current Windows user, call ImGetDWord() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	ImSetSystemDWord()

Install Conduit Manager API

ImGetSystemExtension

ImGetSystemExtension Function

Purpose	Retrieves the file extensions supported by a system-registered install conduit.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI ImGetSystemExtension (DWORD dwID, TCHAR *pExtension, int *piSize)</pre>
Parameters	<p>→ <i>dwID</i> A DWORD that specifies the ID of the install conduit whose Extension value you want to retrieve.</p> <p>← <i>pExtension</i> A pointer to a null-terminated character buffer. Upon return, this is the value of the Extensions configuration entry for the specified install conduit.</p> <p>↔ <i>piSize</i> A pointer to an integer that specifies the size, in TCHARs, of the buffer referenced by the <i>pExtension</i> parameter. Upon return, this is the actual size, in TCHARs, of the Extension string. If this function fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_NO_CONDUIT ERR_STORAGE_ACCESS ERR_BUFFER_TOO_SMALL The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the <i>piSize</i> parameter.</p> <p>ERR_VALUE_NOT_FOUND ERR_INVALID_INSTALL_ID ERR_INVALID_POINTER</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments	This function retrieves the value of the Extensions configuration entry for the specified install conduit. The Extensions string specifies the file extensions that the install conduit supports.
	This function retrieves information about an install conduit that is registered for the system. To retrieve this information for an install conduit that is registered for the current Windows user, call ImGetExtension() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.

See Also [Extensions](#), [ImSetExtension\(\)](#)

Install Conduit Manager API

ImGetSystemMask

ImGetSystemMask Function

Purpose	Retrieves the value of the bit mask for a system-registered install conduit.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI ImGetSystemMask (DWORD dwID, DWORD *pdwMask)</pre>
Parameters	<p>→ <i>dwID</i> A DWORD that specifies the ID of the install conduit whose Mask value you want to retrieve.</p> <p>← <i>pdwMask</i> A pointer to a DWORD. Upon return, this is the value of the Mask configuration entry for the specified install conduit.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_NO_CONDUIT ERR_STORAGE_ACCESS ERR_VALUE_NOT_FOUND ERR_INVALID_INSTALL_ID ERR_INVALID_POINTER</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>
Comments	<p>This function retrieves the value of the Mask configuration entry for the specified install conduit. The Mask is a bit mask value that uniquely identifies an install conduit.</p> <p>This function retrieves information about an install conduit that is registered for the system. To retrieve this information for an install conduit that is registered for the current Windows user, call ImGetMask().</p>
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	Mask , ImSetSystemMask() , “ Registering an Install Conduit ” on page 159 in the <i>C/C++ Sync Suite Companion</i>

ImGetSystemModule Function

Purpose Retrieves the filename of a system-registered install conduit.

Declared In CondMgr.h

Prototype `int WINAPI ImGetSystemModule (DWORD dwID,
 TCHAR *pModule, int *piSize)`

Parameters → *dwID*
A DWORD that specifies the ID of the install conduit whose Module value you want to retrieve.

← *pModule*
A pointer to a null-terminated character buffer. Upon return, this is the value of the [Module](#) configuration entry for the specified conduit.

↔ *piSize*
The size, in TCHARS, of the buffer referenced by the *pModule* parameter. Upon return, this is the actual size, in TCHARS, of the Module string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_NO_CONDUIT

ERR_STORAGE_ACCESS

ERR_BUFFER_TOO_SMALL

The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the *piSize* parameter.

ERR_INVALID_INSTALL_ID

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Install Conduit Manager API

ImGetSystemModule

Comments	This function retrieves the value of the Module configuration entry for the specified install conduit. The Module string is the filename of the install conduit DLL to load when the install conduit is invoked.
	This function retrieves information about an install conduit that is registered for the system. To retrieve this information for an install conduit that is registered for the current Windows user, call _ImGetModule() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.

See Also [Module](#), [_ImSetSystemModule\(\)](#)

ImGetSystemName Function

Purpose Retrieves the display name of a system-registered install conduit.

Declared In CondMgr.h

Prototype int WINAPI ImGetSystemName (DWORD *dwID*,
 TCHAR **pConduitName*, int **piSize*)

Parameters

→ *dwID*

A DWORD that specifies the ID of the install conduit whose Name value you want to retrieve.

← *pConduitName*

A pointer to a null-terminated character buffer. Upon return, this is the value of the [Name](#) configuration entry for the specified install conduit.

↔ *piSize*

A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the *pConduitName* parameter.

Upon return, this is the actual size, in TCHARS, of the Name string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_NO_CONDUIT

ERR_STORAGE_ACCESS

ERR_BUFFER_TOO_SMALL

The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the *piSize* parameter.

ERR_INVALID_INSTALL_ID

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Install Conduit Manager API

ImGetSystemName

Comments	This function retrieves the value of the Name configuration entry for the specified install conduit. The Name string specifies the name of the install conduit to display in HotSync Manager's Custom dialog box.
	This function retrieves information about an install conduit that is registered for the system. To retrieve this information for an install conduit that is registered for the current Windows user, call _ImGetName () .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	Name , _ImSetSystemName ()

ImGetSystemString Function

Purpose Retrieves a string value configuration entry for a system-registered install conduit.

Declared In CondMgr.h

Prototype

```
int WINAPI ImGetSystemString (DWORD dwID,  
    const TCHAR *pValue, TCHAR *pString,  
    int *piSize, const TCHAR *pDefault)
```

Parameters

→ *dwID*
A DWORD that specifies the ID of the install conduit for which you want to retrieve a configuration entry value.

→ *pValue*
The configuration entry name.

← *pString*
A pointer to a character buffer. Upon return, this is the value of the specified configuration entry for the specified conduit.

↔ *piSize*
A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the *pString* parameter. Upon return, this is the actual size, in TCHARS, of the string. If this function fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

→ *pDefault*
The default value to return in **pString* if the specified *pValue* name could not be found in the configuration entries.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_BUFFER_TOO_SMALL

The size of the buffer that you supplied is too small to contain the string, so this function passes back the actual required size into the *piSize* parameter.

ERR_NO_CONDUIT

ERR_STORAGE_ACCESS

ERR_VALUE_NOT_FOUND

Install Conduit Manager API

ImGetSystemString

ERR_INVALID_INSTALL_ID

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments

This function retrieves a string value from the configuration entries for an install conduit. This is a general purpose function for retrieving an install conduit configuration entry value by name. You can use it to retrieve values of configuration entries that you create for your own use with [ImSetSystemString\(\)](#).

If Install Conduit Manager does not find the configuration entry name specified by *pValue*, it passes back the default value (sets **pString* equal to the value of **pDefault*) and returns 0 (success). However, if the install conduit specified by *dwID* does not exist, Install Conduit Manager does *not* pass back the default value but does return an error.

This function retrieves information about an install conduit that is registered for the system. To retrieve this information for an install conduit that is registered for the current Windows user, call [ImGetString\(\)](#).

Compatibility

Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also

[ImSetString\(\)](#)

ImRegister Function

Purpose	Registers an install conduit with HotSync Manager for the current Windows user. Uses a FileInstallType structure.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI ImRegister (const FileInstallType *sConduitInfo)</pre>
Parameters	<p>→ <i>sConduitInfo</i> A FileInstallType structure that specifies the information to register an install conduit.</p>
Returns	<p>If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_CONDUIT_MGR ERR_CONFLICTING_EXTENSION The specified filename extension is already registered for another install conduit. ERR_NO_MEMORY ERR_INSTALL_ID_ALREADY_IN_USE ERR_STORAGE_ACCESS ERR_UNABLE_TO_CREATE_CONDUIT ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_INSTALL_ID</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>
Comments	<p>To register an install conduit, fill in the fields of a FileInstallType structure and pass this function its handle.</p> <p>This function registers an install conduit for the current Windows user. To register an install conduit for the system, call ImRegisterSystem().</p>
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	FileInstallType , ImRegisterID() , ImUnregisterID()

Install Conduit Manager API

ImRegisterID

ImRegisterID Function

Purpose	Registers a new unique ID for an install conduit with HotSync Manager for the current Windows user (requires calling other <code>ImSet...()</code> functions to complete registration).
Declared In	<code>CondMgr.h</code>
Prototype	<code>int WINAPI ImRegisterID (DWORD dwCreatorID)</code>
Parameters	<p><code>→ dwCreatorID</code> A DWORD that specifies the unique ID for the install conduit (the same value as defined in the FileInstallType.dwCreatorID field).</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p><code>ERR_INSTALL_ID_ALREADY_IN_USE</code> <code>ERR_STORAGE_ACCESS</code> <code>ERR_UNABLE_TO_CREATE_CONDUIT</code> <code>ERR_UNABLE_TO_SET_CONDUIT_VALUE</code> <code>ERR_INVALID_INSTALL_ID</code></p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>
Comments	<p>This function creates and sets the CreatorID value for a new set of install conduit configuration entries. To use this function to register an install conduit, you must call this function first; then if this function call succeeds, call other <code>ImSet...()</code> functions to specify the install conduit’s remaining configuration entries.</p> <p>This function sets the creator ID of an install conduit that is not yet registered for the current Windows user. To set this value for an install conduit that is not yet registered for the system, call ImRegisterSystemID().</p>
See Also	FileInstallType , ImRegister() , ImUnregisterID()

ImRegisterSystem Function

Purpose	Registers an install conduit with HotSync Manager for the system. Uses a FileInstallType structure.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI ImRegisterSystem (const FileInstallType *sConduitInfo)</pre>
Parameters	<p>→ <i>sConduitInfo</i> A FileInstallType structure that specifies the information to register an install conduit.</p>
Returns	<p>If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_CONDUIT_MGR ERR_CONFLICTING_EXTENSION The specified filename extension is already registered for another install conduit. ERR_NO_MEMORY ERR_INSTALL_ID_ALREADY_IN_USE ERR_STORAGE_ACCESS ERR_UNABLE_TO_CREATE_CONDUIT ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_INSTALL_ID</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Install Conduit Manager API

ImRegisterSystem

Comments	To register an install conduit, fill in the fields of a FileInstallType structure and pass this function its handle. This function registers an install conduit for the system. To register an install conduit for the current Windows user, call _ImRegister() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	FileInstallType , _ImRegisterSystemID() , _ImUnregisterSystemID()

ImRegisterSystemID Function

Purpose Registers a new unique ID for an install conduit with HotSync Manager for the system (requires calling other `ImSetSystem...()` functions to complete registration).

Declared In CondMgr.h

Prototype int WINAPI ImRegisterSystemID (DWORD dwCreatorID)

Parameters

→ *dwCreatorID*
A DWORD that specifies the unique ID for the install conduit (the same value as defined in the [FileInstallType.dwCreatorID](#) field).

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_INSTALL_ID_ALREADY_IN_USE

ERR_STORAGE_ACCESS

ERR_UNABLE_TO_CREATE_CONDUIT

ERR_UNABLE_TO_SET_CONDUIT_VALUE

ERR_INVALID_INSTALL_ID

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments

This function creates and sets the [CreatorID](#) value for a new set of install conduit configuration entries. To use this function to register an install conduit, you must call this function first; then if this function call succeeds, call other `ImSetSystem...()` functions to specify the install conduit’s remaining configuration entries.

This function sets the creator ID of an install conduit that is not yet registered for the system. To set this value for an install conduit that is not yet registered for the current Windows user, call [ImRegisterID\(\)](#).

Compatibility

Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also

[FileInstallType](#), [ImRegisterSystem\(\)](#),
[ImUnregisterSystemID\(\)](#)

Install Conduit Manager API

ImSetDirectory

ImSetDirectory Function

Purpose	Sets the name of the directory in which a user-registered install conduit looks for files to install.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI ImSetDirectory (DWORD dwID, const TCHAR *pDirectory)</pre>
Parameters	<p>→ <i>dwID</i> A DWORD that specifies the ID of the install conduit whose Directory value you want to set.</p> <p>→ <i>pDirectory</i> A pointer to a null-terminated character buffer specifying the value of the Directory configuration entry for the specified install conduit.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_NO_CONDUIT ERR_STORAGE_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_INSTALL_ID ERR_INVALID_POINTER</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>
Comments	This function sets the value of the Directory configuration entry for the specified install conduit. The directory string specifies the directory, under the current HotSync user’s directory, in which an install conduit looks for files to install. For example, if the value of Directory were Install and the value of Core\Path were a typical value, then the full path to this install conduit’s directory is: <code>C:\Documents and Settings\<WinUsername>\My Documents\Palm OS Desktop\<HotSyncUsername>\Install</code>

This function sets information for an install conduit that is registered for the current Windows user. To set this information for an install conduit that is registered for the system, call [ImSetSystemDirectory\(\)](#).

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [Directory](#), [ImGetDirectory\(\)](#)

Install Conduit Manager API

ImSetDWord

ImSetDWord Function

Purpose	Sets a DWORD configuration entry value for a user-registered install conduit.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI ImSetDWord (DWORD dwID, const TCHAR *pValue, DWORD dwValue)</pre>
Parameters	<p>→ <i>dwID</i> A DWORD that specifies the ID of the install conduit whose value you want to set.</p> <p>→ <i>pValue</i> A pointer to a null-terminated character buffer containing the string name of the configuration entry.</p> <p>→ <i>dwValue</i> The value that you want set for the specified configuration entry.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_NO_CONDUIT ERR_STORAGE_ACCESS ERR_INVALID_INSTALL_ID ERR_INVALID_POINTER ERR_VALUE_NOT_FOUND For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.

Comments	This function sets a DWORD value in the configuration entries for an install conduit. This is a general purpose function for setting an install conduit configuration entry value by name. You can use it to set values of configuration entries that you retrieve for your own use with ImGetDWord() . This function sets information for an install conduit that is registered for the current Windows user. To set this information for an install conduit that is registered for the system, call ImSetSystemDWord() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	ImGetDWord()

Install Conduit Manager API

ImSetExtension

ImSetExtension Function

Purpose	Sets the file extensions supported by a user-registered install conduit.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI ImSetExtension (DWORD dwID, const TCHAR *pExtension)</pre>
Parameters	<p>→ <i>dwID</i> A DWORD that specifies the ID of the install conduit whose Extension value you want to set.</p> <p>→ <i>pExtension</i> A pointer to a null-terminated character buffer specifying the value of the Extensions configuration entry for the specified install conduit.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_NO_CONDUIT ERR_CONFLICTING_EXTENSION The specified filename extension is already registered for another install conduit. ERR_STORAGE_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_INSTALL_ID ERR_INVALID_POINTER For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.

Comments	This function sets the value of the Extensions configuration entry for the specified install conduit. The Extensions string specifies the file extensions that the install conduit supports.
	This function sets information for an install conduit that is registered for the current Windows user. To set this information for an install conduit that is registered for the system, call ImSetSystemExtension() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	ImGetExtension()

Install Conduit Manager API

ImSetMask

ImSetMask Function

Purpose Sets the value of the bit mask for a user-registered install conduit.

Declared In CondMgr.h

Prototype int WINAPI ImSetMask (DWORD *dwID*, DWORD *dwMask*)

Parameters

→ *dwID*
A DWORD that specifies the ID of the install conduit whose mask you want to set.

→ *dwMask*
The DWORD value that you want to set for the [Mask](#) configuration entry for the specified install conduit.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_NO_CONDUIT

ERR_STORAGE_ACCESS

ERR_UNABLE_TO_SET_CONDUIT_VALUE

ERR_INVALID_INSTALL_ID

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments This function sets the value of the [Mask](#) configuration entry for the specified install conduit. The Mask is a bit mask value that uniquely identifies an install conduit.

This function sets information for an install conduit that is registered for the current Windows user. To set this information for an install conduit that is registered for the system, call [ImSetSystemMask\(\)](#).

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [Mask](#), [ImGetMask\(\)](#), “[Registering an Install Conduit](#)” on page 159 in the *C/C++ Sync Suite Companion*

ImSetModule Function

Purpose	Sets the filename of a user-registered install conduit.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI ImSetModule (DWORD dwID, const TCHAR *pModule)</pre>
Parameters	<p>→ <i>dwID</i> A DWORD that specifies the ID of the install conduit whose Module value you want to set.</p> <p>→ <i>pModule</i> A pointer to a null-terminated character buffer specifying the value of the Module configuration entry for the specified install conduit.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_NO_CONDUIT ERR_STORAGE_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_INSTALL_ID ERR_INVALID_POINTER For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	This function sets the value of the Module configuration entry for the specified install conduit. The Module string is the filename of the install conduit DLL to load when the install conduit is invoked. This function sets information for an install conduit that is registered for the current Windows user. To set this information for an install conduit that is registered for the system, call ImSetSystemModule() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	Module , ImGetModule()

Install Conduit Manager API

ImSetName

ImSetName Function

Purpose	Sets the display name of a user-registered install conduit.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI ImSetName (DWORD dwID, const TCHAR *pConduitName)</pre>
Parameters	<p>→ <i>dwID</i> A DWORD that specifies the ID of the install conduit whose Name value you want to set.</p> <p>→ <i>pConduitName</i> A pointer to a null-terminated character buffer specifying the value of the Name configuration entry for the specified install conduit.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_NO_CONDUIT ERR_STORAGE_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_INSTALL_ID ERR_INVALID_POINTER For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	This function sets the value of the Name configuration entry for the specified install conduit. The Name string specifies the name of the install conduit to display in HotSync Manager’s Custom dialog box. This function sets information for an install conduit that is registered for the current Windows user. To set this information for an install conduit that is registered for the system, call ImSetSystemName () .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	Name , ImGetName ()

ImSetString Function

Purpose	Sets a string value configuration entry for a user-registered install conduit.
Declared In	<code>CondMgr.h</code>
Prototype	<pre>int WINAPI ImSetString (DWORD dwID, const TCHAR *pValue, TCHAR *pString)</pre>
Parameters	<p>→ <i>dwID</i> A DWORD that specifies the ID of the install conduit whose value you want to set.</p> <p>→ <i>pValue</i> A pointer to a null-terminated character buffer containing the string name of the configuration entry.</p> <p>→ <i>pString</i> The string value you want to set for the specified configuration entry.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_NO_CONDUIT ERR_STORAGE_ACCESS ERR_INVALID_INSTALL_ID ERR_INVALID_POINTER ERR_VALUE_NOT_FOUND For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.

Install Conduit Manager API

ImSetString

Comments	This function sets a string value in the configuration entries for an install conduit. This is a general purpose function for setting an install conduit configuration entry value by name. You can use it to set values of configuration entries that you retrieve for your own use with ImGetString() . This function sets information for an install conduit that is registered for the current Windows user. To set this information for an install conduit that is registered for the system, call ImSetSystemString() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	ImGetString()

ImSetSystemDirectory Function

Purpose Sets the name of the directory in which a system-registered install conduit looks for files to install.

Declared In CondMgr.h

Prototype

```
int WINAPI ImSetSystemDirectory (DWORD dwID,
                                  const TCHAR *pDirectory)
```

Parameters

→ *dwID*
A DWORD that specifies the ID of the install conduit whose Directory value you want to set.

→ *pDirectory*
A pointer to a null-terminated character buffer specifying the value of the [Directory](#) configuration entry for the specified install conduit.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_NO_CONDUIT
ERR_STORAGE_ACCESS
ERR_UNABLE_TO_SET_CONDUIT_VALUE
ERR_INVALID_INSTALL_ID
ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments This function sets the value of the [Directory](#) configuration entry for the specified install conduit. The directory string specifies the directory, under the current HotSync user’s directory, in which an install conduit looks for files to install. For example, if the value of Directory were Install and the value of [Core\Path](#) were a typical value, then the full path to this install conduit’s directory is:

```
C:\Documents and Settings\<WinUsername>\  
My Documents\Palm OS Desktop\  
<HotSyncUsername>\Install
```

Install Conduit Manager API

ImSetSystemDirectory

This function sets information for an install conduit that is registered for the system. To set this information for an install conduit that is registered for the current Windows user, call [_ImSetDirectory\(\)](#).

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [Directory](#), [_ImGetSystemDirectory\(\)](#)

ImSetSystemDWord Function

Purpose Sets a DWORD configuration entry value for a system-registered install conduit.

Declared In CondMgr.h

Prototype

```
int WINAPI ImSetSystemDWord (DWORD dwID,
                             const TCHAR *pValue, DWORD dwValue)
```

Parameters

→ *dwID*
A DWORD that specifies the ID of the install conduit whose value you want to set.

→ *pValue*
A pointer to a null-terminated character buffer containing the string name of the configuration entry.

→ *dwValue*
The value that you want set for the specified configuration entry.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_NO_CONDUIT
ERR_STORAGE_ACCESS
ERR_INVALID_INSTALL_ID
ERR_INVALID_POINTER
ERR_VALUE_NOT_FOUND

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Install Conduit Manager API

ImSetSystemDWord

Comments	This function sets a DWORD value in the configuration entries for an install conduit. This is a general purpose function for setting an install conduit configuration entry value by name. You can use it to set values of configuration entries that you retrieve for your own use with ImGetDWord() . This function sets information for an install conduit that is registered for the system. To set this information for an install conduit that is registered for the current Windows user, call ImSetDWord() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	ImGetSystemDWord()

ImSetSystemExtension Function

Purpose Sets the file extensions supported by a system-registered install conduit.

Declared In CondMgr.h

Prototype

```
int WINAPI ImSetSystemExtension (DWORD dwID,
                                  const TCHAR *pExtension)
```

Parameters

→ *dwID*
A DWORD that specifies the ID of the install conduit whose Extension value you want to set.

→ *pExtension*
A pointer to a null-terminated character buffer specifying the value of the [Extensions](#) configuration entry for the specified install conduit.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error code values:

ERR_NO_CONDUIT

ERR_CONFLICTING_EXTENSION

The specified filename extension is already registered for another install conduit.

ERR_STORAGE_ACCESS

ERR_UNABLE_TO_SET_CONDUIT_VALUE

ERR_INVALID_INSTALL_ID

ERR_INVALID_POINTER

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Install Conduit Manager API

ImSetSystemExtension

- Comments** This function sets the value of the [Extensions](#) configuration entry for the specified install conduit. The Extensions string specifies the file extensions that the install conduit supports.
- This function sets information for an install conduit that is registered for the system. To set this information for an install conduit that is registered for the current Windows user, call [_ImSetExtension\(\)](#).
- Compatibility** Conduit Manager versions: 3 and later.
Palm OS versions: All.
- See Also** [Extensions](#), [_ImGetSystemExtension\(\)](#)

ImSetSystemMask Function

Purpose	Sets the value of the bit mask for a system-registered install conduit.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI ImSetSystemMask (DWORD dwID, DWORD dwMask)</pre>
Parameters	<p>→ <i>dwID</i> A DWORD that specifies the ID of the install conduit whose mask you want to set.</p> <p>→ <i>dwMask</i> The DWORD value that you want to set for the Mask configuration entry for the specified install conduit.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_NO_CONDUIT ERR_STORAGE_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_INSTALL_ID ERR_INVALID_POINTER</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>
Comments	<p>This function sets the value of the Mask configuration entry for the specified install conduit. The Mask is a bit mask value that uniquely identifies an install conduit.</p> <p>This function sets information for an install conduit that is registered for the system. To set this information for an install conduit that is registered for the current Windows user, call ImSetMask().</p>
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	Mask , ImGetSystemMask() , “Registering an Install Conduit” on page 159 in the <i>C/C++ Sync Suite Companion</i>

Install Conduit Manager API

ImSetSystemModule

ImSetSystemModule Function

Purpose	Sets the filename of a system-registered install conduit.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI ImSetSystemModule (DWORD dwID, const TCHAR *pModule)</pre>
Parameters	<p>→ <i>dwID</i> A DWORD that specifies the ID of the install conduit whose Module value you want to set.</p> <p>→ <i>pModule</i> A pointer to a null-terminated character buffer specifying the value of the Module configuration entry for the specified install conduit.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_NO_CONDUIT ERR_STORAGE_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_INSTALL_ID ERR_INVALID_POINTER For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	This function sets the value of the Module configuration entry for the specified install conduit. The Module string is the filename of the install conduit DLL to load when the install conduit is invoked. This function sets information for an install conduit that is registered for the system. To set this information for an install conduit that is registered for the current Windows user, call ImSetModule() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	Module , ImGetSystemModule()

ImSetSystemName Function

Purpose	Sets the display name of a system-registered install conduit.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI ImSetSystemName (DWORD dwID, const TCHAR *pConduitName)</pre>
Parameters	<p>→ <i>dwID</i> A DWORD that specifies the ID of the install conduit whose Name value you want to set.</p> <p>→ <i>pConduitName</i> A pointer to a null-terminated character buffer specifying the value of the Name configuration entry for the specified install conduit.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_NO_CONDUIT ERR_STORAGE_ACCESS ERR_UNABLE_TO_SET_CONDUIT_VALUE ERR_INVALID_INSTALL_ID ERR_INVALID_POINTER For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	This function sets the value of the Name configuration entry for the specified install conduit. The Name string specifies the name of the install conduit to display in HotSync Manager’s Custom dialog box. This function sets information for an install conduit that is registered for the system. To set this information for an install conduit that is registered for the current Windows user, call ImSetName () .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	Name , ImGetSystemName ()

Install Conduit Manager API

ImSetSystemString

ImSetSystemString Function

Purpose	Sets a string value configuration entry for a system-registered install conduit.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI ImSetSystemString (DWORD dwID, const TCHAR *pValue, TCHAR *pString)</pre>
Parameters	<p>→ <i>dwID</i> A DWORD that specifies the ID of the install conduit whose value you want to set.</p> <p>→ <i>pValue</i> A pointer to a null-terminated character buffer containing the string name of the configuration entry.</p> <p>→ <i>pString</i> The string value you want to set for the specified configuration entry.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values: ERR_NO_CONDUIT ERR_STORAGE_ACCESS ERR_INVALID_INSTALL_ID ERR_INVALID_POINTER ERR_VALUE_NOT_FOUND For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.

Comments	This function sets a string value in the configuration entries for an install conduit. This is a general purpose function for setting an install conduit configuration entry value by name. You can use it to set values of configuration entries that you retrieve for your own use with ImGetString() . This function sets information for an install conduit that is registered for the system. To set this information for an install conduit that is registered for the current Windows user, call ImSetString() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	ImGetSystemString()

Install Conduit Manager API

ImUnregisterID

ImUnregisterID Function

Purpose	Unregisters an install conduit that is registered for the current Windows user.
Declared In	CondMgr.h
Prototype	int WINAPI ImUnregisterID (DWORD <i>dwCreatorID</i>)
Parameters	<p>→ <i>dwCreatorID</i> A DWORD that specifies the ID of the install conduit to be unregistered.</p>
Returns	<p>If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_UNABLE_TO_DELETE ERR_NO_CONDUIT ERR_STORAGE_ACCESS ERR_INVALID_INSTALL_ID</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>
Comments	<p>Given its unique ID, this function unregisters an install conduit—which removes all of its configuration entries.</p> <p>This function unregisters an install conduit for the current Windows user. To unregister an install conduit for the system, call ImUnregisterSystemID().</p>
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	ImRegister() , ImRegisterID()

ImUnregisterSystemID Function

Purpose	Unregisters an install conduit that is registered for the system.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI ImUnregisterSystemID (DWORD dwCreatorID)</pre>
Parameters	<p>→ <i>dwCreatorID</i> A DWORD that specifies the ID of the install conduit to be unregistered.</p>
Returns	<p>If successful, returns 0. If unsuccessful, returns one of the following nonzero error code values:</p> <p>ERR_UNABLE_TO_DELETE ERR_NO_CONDUIT ERR_STORAGE_ACCESS ERR_INVALID_INSTALL_ID</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>
Comments	<p>Given its unique ID, this function unregisters an install conduit—which removes all of its configuration entries.</p> <p>This function unregisters an install conduit for the system. To unregister an install conduit for the current Windows user, call ImUnregisterID().</p>
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	ImRegisterSystem() , ImRegisterSystemID()

Install Conduit Manager API

ImUnregisterSystemID

Notifier Install Manager API

The Notifier Install Manager provides an API for registering a notifier with HotSync® Manager.

The Notifier Install Manager functions are available in CondMgr.dll (which also includes the Conduit Manager functions) and are declared in CondMgr.h.

The sections in this chapter are:

Notifier Path and Filename	909
Notifier Install Manager Functions	910

For more about notifiers, see [Chapter 8, “Writing a Desktop Notifier”](#) on page 93 in the *C/C++ Sync Suite Companion*.

Notifier Path and Filename

Several Notifier Install Manager functions take or pass back a notifier filename. These functions access the configuration entry for the specified notifier, so whatever was written there is what is passed back. For all of these functions, you can specify a fully qualified path. If you specify a filename without a full path, HotSync Manager tries to find the notifier in the directory that HotSync Manager was launched from and then in the directories specified by the Windows PATH environment variable; it does not look in the HotSync Manager core path.

Notifier Install Manager API

Notifier Install Manager Functions

Notifier Install Manager Functions

This section describes the following functions, which allow you to register and manage notifiers.

<u>NmFind()</u>	Retrieves the index of a user-registered notifier specified by filename.
<u>NmFindSystem()</u>	Retrieves the index of a system-registered notifier specified by filename.
<u>NmGetByIndex()</u>	Retrieves the filename of a user-registered notifier specified by index.
<u>NmGetCount()</u>	Retrieves the number of notifiers that are registered for the current Windows user.
<u>NmGetSystemByIndex()</u>	Retrieves the filename of a system-registered notifier specified by index.
<u>NmGetSystemCount()</u>	Retrieves the number of notifiers that are registered for the system.
<u>NmRegister()</u>	Registers a notifier with HotSync Manager for the current Windows user.
<u>NmRegisterSystem()</u>	Registers a notifier with HotSync Manager for the system.
<u>NmRenameByIndex()</u>	Changes the filename of a user-registered notifier specified by index.
<u>NmRenameSystemByIndex()</u>	Changes the filename of a system-registered notifier specified by index.
<u>NmUnregister()</u>	Unregisters a notifier that is registered for the current Windows user.
<u>NmUnregisterSystem()</u>	Unregisters a notifier that is registered for the system.

NmFind Function

Purpose Retrieves the index of a user-registered notifier specified by filename.

Declared In CondMgr.h

Prototype int WINAPI NmFind (const TCHAR **pNotifier*)

Parameters → *pNotifier*

A const pointer to a null-terminated character buffer containing the filename of the notifier (see “[Notifier Path and Filename](#)” on page 909).

Returns If successful, returns the index of the specified notifier ($>=0$).

If unsuccessful, returns one of the following negative error code values:

ERR_INVALID_POINTER

ERR_NOTIFIER_NOT_FOUND

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Comments The notifier index is maintained by the Notifier Install Manager. Your application does not need to keep track of or modify these indices; the Notifier Install Manager handles this for you.

This function retrieves information about a notifier that is registered for the current Windows user. To retrieve this information for a notifier that is registered for the system, call [NmFindSystem\(\)](#).

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [NmGetByIndex\(\)](#)

Notifier Install Manager API

NmFindSystem

NmFindSystem Function

Purpose	Retrieves the index of a system-registered notifier specified by filename.
Declared In	CondMgr.h
Prototype	int WINAPI NmFindSystem (const TCHAR * <i>pNotifier</i>)
Parameters	<i>pNotifier</i> A const pointer to a null-terminated character buffer containing the filename of the notifier (see “ Notifier Path and Filename ” on page 909).
Returns	If successful, returns the index of the specified notifier (≥ 0). If unsuccessful, returns one of the following negative error code values: ERR_INVALID_POINTER ERR_NOTIFIER_NOT_FOUND For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	The notifier index is maintained by the Notifier Install Manager. Your application does not need to keep track of or modify these indices; the Notifier Install Manager handles this for you. This function retrieves information about a notifier that is registered for the system. To retrieve this information for a notifier that is registered for the current Windows user, call NmFind() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	NmGetSystemByIndex()

NmGetByIndex Function

Purpose Retrieves the filename of a user-registered notifier specified by index.

Declared In CondMgr.h

Prototype

```
int WINAPI NmGetByIndex (int iIndex,  
                         TCHAR *pNotifier, int *piSize)
```

Parameters

→ *iIndex*

The zero-based index of the notifier you want to retrieve.

← *pNotifier*

A pointer to a null-terminated character buffer specifying the filename of the notifier (see “[Notifier Path and Filename](#)” on page 909).

↔ *piSize*

A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the *pNotifier* parameter. Upon return, this is the actual size, in TCHARS, of the string. If conversion fails because the string buffer is too small, the value of *piSize* upon return is the required size of the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following negative error code values:

ERR_INVALID_POINTER

ERR_BUFFER_TOO_SMALL

The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the *piSize* parameter.

ERR_INDEX_OUT_OF_RANGE

ERR_NOTIFIER_NOT_FOUND

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Notifier Install Manager API

NmGetByIndex

Comments	This function retrieves the filename a notifier, which is stored in the HotSync Manager portion of the configuration entries. This function retrieves information about a notifier that is registered for the current Windows user. To retrieve this information for a notifier that is registered for the system, call NmGetSystemByIndex() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	NmGetCount() , NmFind() , NmRenameByIndex()

NmGetCount Function

Purpose	Retrieves the number of notifiers that are registered for the current Windows user.
Declared In	CondMgr.h
Prototype	int WINAPI NmGetCount (DWORD *pdwCount)
Parameters	$\leftarrow pdwCount$ A pointer to a DWORD that is set to the number of registered notifiers.
Returns	If successful, returns 0. If unsuccessful, returns one of the following negative error code values: ERR_NOTIFIER_NOT_FOUND ERR_STORAGE_ACCESS For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	This function retrieves information about notifiers that are registered for the current Windows user. To retrieve this information for notifiers that are registered for the system, call NmGetSystemCount () .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	NmGetByIndex ()

Notifier Install Manager API

NmGetSystemByIndex

NmGetSystemByIndex Function

Purpose	Retrieves the filename of a system-registered notifier specified by index.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI NmGetSystemByIndex (int iIndex, TCHAR *pNotifier, int *piSize)</pre>
Parameters	<p>→ <i>iIndex</i> The zero-based index of the notifier you want to retrieve.</p> <p>← <i>pNotifier</i> A pointer to a null-terminated character buffer specifying the filename of the notifier (see “Notifier Path and Filename” on page 909).</p> <p>↔ <i>piSize</i> A pointer to an integer that specifies the size, in TCHARS, of the buffer referenced by the <i>pNotifier</i> parameter. Upon return, this is the actual size, in TCHARS, of the string. If conversion fails because the string buffer is too small, the value of <i>piSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following negative error code values:</p> <p>ERR_INVALID_POINTER ERR_BUFFER_TOO_SMALL The size of the buffer that you supplied is too small to contain the string, so this function passes back the required size in the <i>piSize</i> parameter.</p> <p>ERR_INDEX_OUT_OF_RANGE ERR_NOTIFIER_NOT_FOUND</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>

Comments This function retrieves the filename a notifier, which is stored in the HotSync Manager portion of the configuration entries.

This function retrieves information about a notifier that is registered for the system. To retrieve this information for a notifier that is registered for the current Windows user, call [NmGetByIndex\(\)](#).

Compatibility Conduit Manager versions: 3 and later.

Palm OS versions: All.

See Also [NmGetSystemCount\(\)](#), [NmFindSystem\(\)](#),
[NmRenameSystemByIndex\(\)](#)

Notifier Install Manager API

NmGetSystemCount

NmGetSystemCount Function

Purpose	Retrieves the number of notifiers that are registered for the system.
Declared In	CondMgr.h
Prototype	int WINAPI NmGetSystemCount (DWORD *pdwCount)
Parameters	$\leftarrow pdwCount$ A pointer to a DWORD that is set to the number of registered notifiers.
Returns	If successful, returns 0. If unsuccessful, returns one of the following negative error code values: ERR_NOTIFIER_NOT_FOUND ERR_STORAGE_ACCESS For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	This function retrieves information about notifiers that are registered for the system. To retrieve this information for notifiers that are registered for the current Windows user, call NmGetCount () .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	NmGetSystemByIndex ()

NmRegister Function

Purpose	Registers a notifier with HotSync Manager for the current Windows user.
Declared In	CondMgr.h
Prototype	int WINAPI NmRegister (const TCHAR * <i>pNotifierPath</i>)
Parameters	<i>→ pNotifierPath</i> A pointer to a null-terminated character buffer specifying the filename of the notifier to be registered with HotSync Manager (see “ Notifier Path and Filename ” on page 909).
Returns	If successful, returns 0. If unsuccessful, returns one of the following negative error code values: ERR_ALREADY_INSTALLED ERR_INVALID_POINTER ERR_STORAGE_ACCESS For more information about the error codes, see “ Conduit Manager Error Codes ” on page 843.
Comments	This function registers a notifier for the current Windows user, storing its path in the configuration entries. To register a notifier for the system, call NmRegisterSystem() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	NmUnregister()

Notifier Install Manager API

NmRegisterSystem

NmRegisterSystem Function

Purpose	Registers a notifier with HotSync Manager for the system.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI NmRegisterSystem (const TCHAR *pNotifierPath)</pre>
Parameters	<p><i>→ pNotifierPath</i></p> <p>A pointer to a null-terminated character buffer specifying the filename of the notifier to be registered with HotSync Manager (see “Notifier Path and Filename” on page 909).</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following negative error code values:</p> <p>ERR_ALREADY_INSTALLED ERR_INVALID_POINTER ERR_STORAGE_ACCESS</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>
Comments	This function registers a notifier for the system, storing its path in the configuration entries. To register a notifier for the current Windows user, call NmRegister() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	NmUnregisterSystem()

NmRenameByIndex Function

Purpose Changes the filename of a user-registered notifier specified by index.

Declared In CondMgr.h

Prototype

```
int WINAPI NmRenameByIndex (int iIndex,
                           const TCHAR *pNotifier)
```

Parameters

→ *iIndex*
The zero-based index of the notifier you want to rename.
Note that this index is *not* based on the configuration entry names—that is, Notifier0, Notifier1, and so on. Rather it is an independent index maintained by the Notifier Install Manager.

→ *pNotifier*

A pointer to a null-terminated character buffer specifying the filename of the notifier (see “[Notifier Path and Filename](#)” on page 909).

Returns If successful, returns 0.

If unsuccessful, returns one of the following negative error code values:

ERR_INVALID_POINTER

ERR_INVALID_PATH

ERR_INDEX_OUT_OF_RANGE

ERR_NOTIFIER_NOT_FOUND

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Notifier Install Manager API

NmRenameByIndex

Comments This function changes the filename of a notifier that is stored as a value in the HotSync Manager portion of the configuration entries.

IMPORTANT: Do not set *pNotifier* to NULL as a way of deleting a notifier. HotSync Manager displays an error message when it tries to call such a notifier. Use the [NmUnregister\(\)](#) function instead.

This function sets information for a notifier that is registered for the current Windows user. To set this information for a notifier that is registered for the system, call [NmRenameSystemByIndex\(\)](#).

Compatibility Conduit Manager versions: 2 and later.
Palm OS versions: All.

See Also [NmUnregister\(\)](#)

NmRenameSystemByIndex Function

Purpose Changes the filename of a system-registered notifier specified by index.

Declared In CondMgr.h

Prototype

```
int WINAPI NmRenameSystemByIndex (int iIndex,  
                                  const TCHAR *pNotifier)
```

Parameters

→ *iIndex*
The zero-based index of the notifier you want to rename.
Note that this index is *not* based on the configuration entry names—that is, Notifier0, Notifier1, and so on. Rather it is an independent index maintained by the Notifier Install Manager.

→ *pNotifier*

A pointer to a null-terminated character buffer specifying the filename of the notifier (see “[Notifier Path and Filename](#)” on page 909).

Returns If successful, returns 0.

If unsuccessful, returns one of the following negative error code values:

ERR_INVALID_POINTER

ERR_INVALID_PATH

ERR_INDEX_OUT_OF_RANGE

ERR_NOTIFIER_NOT_FOUND

For more information about the error codes, see “[Conduit Manager Error Codes](#)” on page 843.

Notifier Install Manager API

NmRenameSystemByIndex

Comments This function changes the filename of a notifier that is stored as a value in the HotSync Manager portion of the configuration entries.

IMPORTANT: Do not set *pNotifier* to NULL as a way of deleting a notifier. HotSync Manager displays an error message when it tries to call such a notifier. Use the [NmUnregisterSystem\(\)](#) function instead.

This function sets information for a notifier that is registered for the system. To set this information for a notifier that is registered for the current Windows user, call [NmRenameByIndex\(\)](#).

Compatibility Conduit Manager versions: 3 and later.
Palm OS versions: All.

See Also [NmUnregisterSystem\(\)](#)

NmUnregister Function

Purpose	Unregisters a notifier that is registered for the current Windows user.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI NmUnregister (const TCHAR *pNotifierPath)</pre>
Parameters	<p>→ <i>pNotifierPath</i> A pointer to a null-terminated character buffer specifying the filename of the notifier to be unregistered with HotSync Manager (see “Notifier Path and Filename” on page 909).</p>
Returns	<p>If successful, returns 0. If unsuccessful, returns one of the following negative error code values:</p> <p>ERR_INVALID_POINTER ERR_NOTIFIER_NOT_FOUND ERR_STORAGE_ACCESS</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>
Comments	This function unregisters a notifier that is registered for the current Windows user, removing its path from the configuration entries but not deleting the notifier DLL itself. To unregister a notifier for the system, call NmUnregisterSystem() .
Compatibility	Conduit Manager versions: 2 and later. Palm OS versions: All.
See Also	NmRegister()

Notifier Install Manager API

NmUnregisterSystem

NmUnregisterSystem Function

Purpose	Unregisters a notifier that is registered for the system.
Declared In	CondMgr.h
Prototype	<pre>int WINAPI NmUnregisterSystem (const TCHAR *pNotifierPath)</pre>
Parameters	<p>→ <i>pNotifierPath</i> A pointer to a null-terminated character buffer specifying the filename of the notifier to be unregistered with HotSync Manager (see “Notifier Path and Filename” on page 909).</p>
Returns	<p>If successful, returns 0. If unsuccessful, returns one of the following negative error code values:</p> <p>ERR_INVALID_POINTER ERR_NOTIFIER_NOT_FOUND ERR_STORAGE_ACCESS</p> <p>For more information about the error codes, see “Conduit Manager Error Codes” on page 843.</p>
Comments	This function unregisters a notifier that is registered for the system, removing its path from the configuration entries but not deleting the notifier DLL itself. To unregister a notifier for the current Windows user, call NmUnregister() .
Compatibility	Conduit Manager versions: 3 and later. Palm OS versions: All.
See Also	NmRegisterSystem()

HotSync Manager API

Use the HotSync® Manager API to enable your application to control HotSync Manager in the following ways:

- Stop, start, restart, or refresh HotSync Manager.
- Call HotSync Manager dialogs to customize conduits or change connection settings or to display the HotSync Log.
- Check or set the connection status type for serial, modem, infrared, USB, and network connections.
- Check whether HotSync Manager is idle or synchronizing.

The HotSync Manager API functions are available in HSAPI.dll and declared in HSAPI.h.

The sections in this chapter are:

HotSync Manager API Constants	928
HotSync Manager API Functions	935
HotSync Manager API Error Codes	949

HotSync Manager API

HotSync Manager API Constants

HotSync Manager API Constants

This section describes the following enumerated constants and groups of preprocessor constants that you use with the HotSync Manager API functions.

<u>Connection Status Flags</u>	Indicate whether a connection type is enabled or disabled.
<u>HotSync Manager API Versions</u>	Defines the HotSync Manager API version numbers that <u>HsGetApiVersion()</u> can pass back.
<u>HSConnectionType</u>	Defines the types of connection that HotSync Manager can make with the handheld.
<u>HsStatusType</u>	Specifies the actions that <u>HsSetAppStatus()</u> can perform.
<u>Start Options</u>	Specify options for starting the HotSync Manager application with <u>HsSetAppStatus()</u> .
<u>Synchronization Status Flags</u>	Indicate whether HotSync Manager is performing a HotSync operation.

Connection Status Flags

Purpose	Indicate whether a connection type is enabled or disabled.
Declared In	HSAPI.h
Constants	#define HSCONNECTION_DISABLED 0x00000000 Disable the specified connection type. #define HSCONNECTION_ENABLED 0x00000001 Enable the specified connection type.
Compatibility	HotSync Manager API version: All. Palm OS version: All.
See Also	HsGetCommStatus() , HsSetCommStatus()

HotSync Manager API Versions

Purpose	Defines the HotSync Manager API version numbers that HsGetApiVersion() can pass back.
Declared In	HSAPI.h
Constants	#define HS_API_VERSION_1 0x00010001 The value returned by HotSync Manager API version 1. #define HS_API_VERSION_2 0x00010002 The value returned by HotSync Manager API version 2.
Comments	Note that the HotSync Manager API version number is not necessarily the same as that of the HotSync Manager application.
Compatibility	HotSync Manager API version: All. Palm OS version: All.
See Also	HsGetApiVersion()

HotSync Manager API

HSConnectionType

HSConnectionType Enum

Purpose Defines the types of connection that HotSync Manager can make with the handheld.

Declared In HSAPI.h

Constants CtSerialPort = 0
Local (serial) connection method.

CtModemPort = 1
Modem connection method.

CtNetwork = 2
Network connection method.

CtIr = 3
Infrared connection method.

CtUSB = 4
Local USB connection method.

CtReserved
Reserved for future use.

Compatibility HotSync Manager API version: All.
Palm OS version: All.

See Also [HsGetCommStatus\(\)](#), [HsSetCommStatus\(\)](#)

HsStatusType Enum

Purpose	Specifies the actions that HsSetAppStatus () can perform.								
Declared In	<code>HSAPI.h</code>								
Constants	<table><tr><td><code>HsCloseApp</code></td><td>Close the HotSync Manager application, if it is running.</td></tr><tr><td><code>HsStartApp</code></td><td>Start the HotSync Manager application, if it is not already running.</td></tr><tr><td><code>HsRestart</code></td><td>Close and restart the HotSync Manager application.</td></tr><tr><td><code>HsReserved</code></td><td>Reserved for future use.</td></tr></table>	<code>HsCloseApp</code>	Close the HotSync Manager application, if it is running.	<code>HsStartApp</code>	Start the HotSync Manager application, if it is not already running.	<code>HsRestart</code>	Close and restart the HotSync Manager application.	<code>HsReserved</code>	Reserved for future use.
<code>HsCloseApp</code>	Close the HotSync Manager application, if it is running.								
<code>HsStartApp</code>	Start the HotSync Manager application, if it is not already running.								
<code>HsRestart</code>	Close and restart the HotSync Manager application.								
<code>HsReserved</code>	Reserved for future use.								
Compatibility	HotSync Manager API version: All. Palm OS version: All.								
See Also	HsSetAppStatus ()								

HotSync Manager API

Start Options

Start Options

Purpose Specify options for starting the HotSync Manager application with [HsSetAppStatus\(\)](#).

Declared In HSAPI.h

Constants

```
#define HSFLAG_DEVICE_SYNC_CHECK 0x00010000
Run HotSync Manager in start/stop sync mode. In this
mode, during a HotSync operation with the handheld,
HotSync Manager runs no conduits; it only validates the
connection. This is the same as starting HotSync Manager
from the command line with the -c option.

#define HSFLAG_INSPECT_CONDUIT 0x00001000
Run HotSync Manager and launch the Conduit Inspector
utility. This is the same as starting HotSync Manager from the
command line with the -ic option. For more information on
Conduit Inspector, see Chapter 6, “Conduit Inspector
Utility,” on page 29 in the Conduit Development Utilities Guide.
This value is defined only for HotSync Manager API versions
2 and later.

#define HSFLAG_LOG_DEBUG_LEVEL_1 0x00000100
Run HotSync Manager in debug log mode 1. This is the same
as starting HotSync Manager from the command line with
the -L1 option.

#define HSFLAG_LOG_DEBUG_LEVEL_2 0x00000200
Run HotSync Manager in debug log mode 2. This is the same
as starting HotSync Manager from the command line with
the -L2 option.

#define HSFLAG_NONE 0x00000000
Set no flags. This is the same as starting HotSync Manager
from the command line with no options.

#define HSFLAG_RESTORE_REGISTRY 0x00000001
Restore any missing configuration entries. This is the same as
starting HotSync Manager from the command line with the
-r option.

#define HSFLAG_RESTORE_REGISTRY_DEFAULT 0x00000002
Restore configuration entries to defaults. This is the same as
starting HotSync Manager from the command line with the
-d option.
```

```
#define HSFLAG_VERBOSE 0x00000010
```

Run HotSync Manager in verbose log mode. This is the same as starting HotSync Manager from the command line with the -v option.

Comments Each of these options corresponds to a HotSync Manager command-line option described in more detail in “[Using Command-line Options for HotSync Manager](#)” on page 24 in *Conduit Development Utilities Guide*.

Compatibility HotSync Manager API version: All, except that HSFLAG_INSPECT_CONDUIT is defined only for versions 2 and later.
Palm OS version: All.

See Also [HsSetAppStatus\(\)](#)

Synchronization Status Flags

Purpose	Indicate whether HotSync Manager is performing a HotSync operation.
Declared In	HSAPI.h
Constants	<pre>#define HOTSYNC_STATUS_IDLE 0 HotSync Manager is not performing a HotSync operation. #define HOTSYNC_STATUS_SYNCING 1 HotSync Manager is performing a HotSync operation.</pre>
Compatibility	HotSync Manager API version: All. Palm OS version: All.
See Also	HsGetSyncStatus ()

HotSync Manager API Functions

This section describes the following functions, which an application can use to control the HotSync Manager application.

<u>HsCheckApiStatus()</u>	Checks whether the API can communicate with the HotSync Manager application.
<u>HsDisplayCustomDlg()</u>	Displays the HotSync Manager application's Custom dialog box.
<u>HsDisplayFileLink()</u>	(<i>Deprecated</i>) Displays the HotSync Manager application's File Link wizard.
<u>HsDisplayLog()</u>	Displays the HotSync Manager application's HotSync Log window.
<u>HsDisplaySetupDlg()</u>	Displays the HotSync Manager application's Setup dialog box.
<u>HsGetApiVersion()</u>	Retrieves the version of the HotSync Manager API.
<u>HsGetCommStatus()</u>	Determines whether a HotSync Manager connection type is enabled.
<u>HsGetSyncStatus()</u>	Determines whether the HotSync Manager application is currently busy synchronizing a handheld.
<u>HsRefreshConduitInfo()</u>	Requests the HotSync Manager application to reload information about all registered conduits.
<u>HsResetComm()</u>	Re-initializes the enabled HotSync Manager communications transports.
<u>HsSetAppStatus()</u>	Closes, starts, or restarts the HotSync Manager application.
<u>HsSetCommStatus()</u>	Sets the status of the HotSync Manager application's communication types.

HotSync Manager API

HsCheckApiStatus

HsCheckApiStatus Function

Purpose	Checks whether the API can communicate with the HotSync Manager application.
Declared In	<code>HSAPI.h</code>
Prototype	<code>long HsCheckApiStatus (void)</code>
Parameters	None.
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error codes. <code>ERROR_HSAPI_HOTSYNC_NOT_FOUND</code> <code>ERROR_HSAPI_FAILURE</code> For descriptions of all HotSync Manager API error codes, see “ HotSync Manager API Error Codes ” on page 949.
Comments	Call this function to determine whether your application can communicate with HotSync Manager application before calling other HotSync Manager API functions.
Compatibility	HotSync Manager API version: All. Palm OS version: All.

HsDisplayCustomDlg Function

Purpose	Displays the HotSync Manager application's Custom dialog box.
Declared In	HSAPI.h
Prototype	long HsDisplayCustomDlg (void)
Parameters	None.
Returns	If successful, returns 0. If unsuccessful, returns the following nonzero error code. ERROR_HSAPI_HOTSYNC_NOT_FOUND For descriptions of all HotSync Manager API error codes, see " HotSync Manager API Error Codes " on page 949.
Comments	The Custom dialog box enables the user to view or change the synchronization preferences of each conduit—for example, Synchronize, Desktop overwrites handheld, and so on.
Compatibility	HotSync Manager API version: All. Palm OS version: All.
See Also	HsDisplaySetupDlg() , HsDisplayLog() , HsDisplayFileLink()

HotSync Manager API

HsDisplayFileLink

HsDisplayFileLink Function

Purpose	(<i>Degraded</i>) Displays the HotSync Manager application's File Link wizard.
Declared In	<code>HSAPI.h</code>
Prototype	<code>long HsDisplayFileLink (DWORD dwUserId)</code>
Parameters	$\rightarrow dwUserId$ The user ID, which specifies whose file linking to set up. If this value is 0, the current user's file link information is displayed.
Returns	If successful, returns 0. If unsuccessful, returns the following nonzero error code. <code>ERROR_HSAPI_HOTSYNC_NOT_FOUND</code> For descriptions of all HotSync Manager API error codes, see " "HotSync Manager API Error Codes" " on page 949.
Comments	The File Link wizard enables the user to create or modify a file link .
Compatibility	HotSync Manager API version: All. Palm OS version: All. This function is deprecated because the file link feature has been removed from HotSync Manager versions 6.0.1 and later.
See Also	HsDisplayCustomDlg() , HsDisplaySetupDlg() , HsDisplayLog()

HsDisplayLog Function

Purpose	Displays the HotSync Manager application's HotSync Log window.
Declared In	HSAPI.h
Prototype	long HsDisplayLog (DWORD dwUserId)
Parameters	$\rightarrow dwUserId$ The user ID, which specifies whose log information to display. If this value is 0, the current user's log is displayed.
Returns	If successful, returns 0. If unsuccessful, returns the following nonzero error code. ERROR_HSAPI_HOTSYNC_NOT_FOUND For descriptions of all HotSync Manager API error codes, see " HotSync Manager API Error Codes " on page 949.
Comments	This function displays the HotSync Log dialog box, which enables the user to view log entries written to it during previous HotSync operations.
Compatibility	HotSync Manager API version: All. Palm OS version: All.
See Also	HsDisplayCustomDlg() , HsDisplaySetupDlg() , HsDisplayFileLink()

HotSync Manager API

HsDisplaySetupDlg

HsDisplaySetupDlg Function

Purpose	Displays the HotSync Manager application's Setup dialog box.
Declared In	HSAPI.h
Prototype	long HsDisplaySetupDlg (void)
Parameters	None.
Returns	If successful, returns 0. If unsuccessful, returns the following nonzero error code. <code>ERROR_HSAPI_HOTSYNC_NOT_FOUND</code> For descriptions of all HotSync Manager API error codes, see " HotSync Manager API Error Codes " on page 949.
Comments	The Setup dialog box enables the user to select a serial port, configure a modem, and set up network HotSync operation for each user.
Compatibility	HotSync Manager API version: All. Palm OS version: All.
See Also	HsDisplayCustomDlg() , HsDisplayLog() , HsDisplayFileLink()

HsGetApiVersion Function

Purpose	Retrieves the version of the HotSync Manager API.
Declared In	HSAPI.h
Prototype	long HsGetApiVersion (DWORD *pdwVersion)
Parameters	\leftrightarrow pdwVersion A pointer to a DWORD to receive the version of the API. HotSync Manager returns one of the values defined in "HotSync Manager API Versions" on page 929.
Returns	If successful, returns 0. If unsuccessful, returns the following nonzero error code. ERROR_HSAPI_INVALID_POINTER For descriptions of all HotSync Manager API error codes, see "HotSync Manager API Error Codes" on page 949.
Comments	Note that the HotSync Manager API version number is not necessarily the same as that of the HotSync Manager application.
Compatibility	HotSync Manager API version: All. Palm OS version: All.
See Also	"HotSync Manager API Versions" on page 929

HotSync Manager API

HsGetCommStatus

HsGetCommStatus Function

Purpose	Determines whether a HotSync Manager connection type is enabled.
Declared In	<code>HSAPI.h</code>
Prototype	<code>long HsGetCommStatus (HSConnectionType connType, DWORD *pdwStatus)</code>
Parameters	<p>→ <i>connType</i> The communication type to retrieve the status of. Specify one of the HSConnectionType enum values.</p> <p>← <i>pdwStatus</i> A pointer to one of the values defined in “Connection Status Flags” on page 929, which indicates whether the connection type is enabled.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns the following nonzero error code.</p> <p><code>ERROR_HSAPI_INVALID_CONN_TYPE</code></p> <p>For descriptions of all HotSync Manager API error codes, see “HotSync Manager API Error Codes” on page 949.</p>
Comments	HotSync Manager connections types include serial, modem, infrared, network, and USB.
Compatibility	HotSync Manager API version: All. Palm OS version: All.
See Also	HsResetComm() , HsSetCommStatus()

HsGetSyncStatus Function

Purpose Determines whether the HotSync Manager application is currently busy synchronizing a handheld.

Declared In HSAPI.h

Prototype long HsGetSyncStatus (DWORD **pdwStatus*)

Parameters ↔ *pdwStatus*

A pointer to one of the status values defined in “[Synchronization Status Flags](#)” on page 934.

Returns If successful, returns 0.

If unsuccessful, returns the following nonzero error code.

ERROR_HSAPI_HOTSYNC_NOT_FOUND

ERROR_HSAPI_INVALID_POINTER

For descriptions of all HotSync Manager API error codes, see “[HotSync Manager API Error Codes](#)” on page 949.

Compatibility HotSync Manager API version: All.
Palm OS version: All.

See Also “[Synchronization Status Flags](#)” on page 934

HotSync Manager API

HsRefreshConduitInfo

HsRefreshConduitInfo Function

Purpose	Requests the HotSync Manager application to reload information about all registered conduits.
Declared In	HSAPI.h
Prototype	long HsRefreshConduitInfo (void)
Parameters	None.
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error codes. ERROR_HSAPI_HOTSYNC_NOT_FOUND For descriptions of all HotSync Manager API error codes, see " HotSync Manager API Error Codes " on page 949.
Comments	HotSync Manager versions <i>earlier than 6.0</i> must be refreshed or restarted after registering a conduit. <i>Versions 6.0 and later</i> automatically refresh their lists so that calling this function is unnecessary. If you register a conduit while HotSync Manager is running, your installer can call this function to make HotSync Manager reload the conduit configuration entries and recognize your newly registered conduit. If your installer changes other settings not related to a conduit (HotSync Manager communication settings, backup conduit, and so on), this function does not reload those settings; use HsSetAppStatus () instead.
Compatibility	HotSync Manager API version: All. Palm OS version: All.
See Also	HsSetAppStatus ()

HsResetComm Function

Purpose	Re-initializes the enabled HotSync Manager communications transports.
Declared In	HSAPI.h
Prototype	long HsResetComm (void)
Parameters	None.
Returns	If successful, returns 0. If unsuccessful, returns the following nonzero error code. ERROR_HSAPI_HOTSYNC_NOT_FOUND For descriptions of all HotSync Manager API error codes, see " HotSync Manager API Error Codes " on page 949.
Comments	Calling this function re-initializes only those HotSync Manager communications transports that are enabled. It doesn't affect disabled transports nor does it change the enabled/disabled status of any transports. If you use only the other HotSync Manager APIs to make transport changes, then you do not need to call this function.
Compatibility	HotSync Manager API version: All. Palm OS version: All.
See Also	HsGetCommStatus() , HsSetCommStatus() , HsSetAppStatus()

HotSync Manager API

HsSetAppStatus

HsSetAppStatus Function

Purpose Closes, starts, or restarts the HotSync Manager application.

Declared In HSAPI.h

Prototype long HsSetAppStatus (HsStatusType *statusType*,
 DWORD *dwStartFlags*)

Parameters → *statusType*
Specify a [HsStatusType](#) enum value to indicate whether to start, stop, or restart HotSync Manager.

→ *dwStartFlags*
If starting or restarting HotSync Manager, specify a valid combination of start options as defined in “[Start Options](#)” on page 932.

Returns If successful, returns 0.

If unsuccessful, returns one of the following nonzero error codes.

ERROR_HSAPI_UNABLE_TO_CLOSE

ERROR_HSAPI_UNKNOWN_STATUS_TYPE

ERROR_HSAPI_UNABLE_TO_START

For descriptions of all HotSync Manager API error codes, see “[HotSync Manager API Error Codes](#)” on page 949.

Comments If you set the *statusType* parameter to HsRestart and HotSync Manager is already running, this function closes and starts HotSync Manager with the options passed in via the *dwStartFlags* flags.

NOTE: Whenever you change any of the configuration entries, HotSync Manager versions earlier than 6.0 require that you call either [HsSetAppStatus\(\)](#) to restart HotSync Manager to recognize all but conduit configuration changes or [HsRefreshConduitInfo\(\)](#) to recognize a conduit configuration change.

However, HotSync Manager versions 6.0 and later automatically discover changes to conduit configuration information without you calling these functions. To recognize other changes (HotSync Manager communication settings, backup conduit, and so on), you must still call [HsSetAppStatus\(\)](#) to restart HotSync Manager.

If you set the *statusType* parameter to HsStartApp and HotSync Manager is already running, this function ignores the *dwStartFlags* flags and generates no error.

If you set the *statusType* parameter to HsCloseApp and HotSync Manager is not running, this function generates no error.

Compatibility
HotSync Manager API version: All.
Palm OS version: All.

See Also [HsResetComm\(\)](#), [HsRefreshConduitInfo\(\)](#)

HotSync Manager API

HsSetCommStatus

HsSetCommStatus Function

Purpose	Sets the status of the HotSync Manager application's communication types.
Declared In	<code>HSAPI.h</code>
Prototype	<code>long HsSetCommStatus (HSConnectionType connType, DWORD dwStatus)</code>
Parameters	<p>→ <i>connType</i> The communication type to set the status of. Specify one of the HSConnectionType enum values.</p> <p>→ <i>dwStatus</i> Enable or disable a communication type. Specify one of the values defined in “Connection Status Flags” on page 929.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following nonzero error codes. <code>ERROR_HSAPI_INVALID_STATUS_FLAG</code> <code>ERROR_HSAPI_INVALID_CONN_TYPE</code> <code>ERROR_HSAPI_FAILURE</code> <code>ERROR_HSAPI_REG_FAILURE</code> For descriptions of all HotSync Manager API error codes, see “ HotSync Manager API Error Codes ” on page 949.
Comments	When HotSync Manager is running, this method requests that it change the status of the specified communication type. If HotSync Manager is not running, this method configures its stored preferences.
Compatibility	HotSync Manager API version: All. Palm OS version: All.
See Also	HsResetComm() , HsGetCommStatus()

HotSync Manager API Error Codes

[Table 14.1](#) lists the values of error codes that HotSync Manager API functions can return. The description of each function states which errors each function can return.

All of the named error codes below are defined as preprocessor constants, which are declared in the HSAPI.h header file.

Table 14.1 HotSync Manager API error codes

Value	Code	Description
-1	—	A nonspecific error occurred.
0x10000000	ERROR_HSAPI_ERROR_BASE	No function returns this value. It is only the offset from which all other error codes are based.
0x10000001	ERROR_HSAPI_HOTSYNC_NOT_FOUND	HotSync Manager is not running.
0x10000002	ERROR_HSAPI_INVALID_CONN_TYPE	The specified connection type is not one defined by the HSConnectionType enum.
0x10000003	ERROR_HSAPI_FAILURE	The API cannot communicate with HotSync Manager.
0x10000004	ERROR_HSAPI_REG_FAILURE	The API cannot access configuration entries.
0x10000005	ERROR_HSAPI_UNKNOWN_STATUS_TYPE	The specified status type is not one defined by the HsStatusType enum.
0x10000006	ERROR_HSAPI_UNABLE_TO_CLOSE	The API cannot close the HotSync Manager application.

HotSync Manager API

HotSync Manager API Error Codes

Table 14.1 HotSync Manager API error codes (*continued*)

Value	Code	Description
0x10000007	ERROR_HSAPI_NO_HOTSYNC_PATH	The HotSync Manager path could not be found.
0x10000008	ERROR_HSAPI_UNABLE_TO_START	The API cannot start the HotSync Manager application.
0x10000009	ERROR_HSAPI_INVALID_STATUS_FLAG	The specified connection type status is not one defined in the “ Connection Status Flags ” on page 929.
0x10000010	ERROR_HSAPI_INVALID_POINTER	The pointer passed into this function is not valid.

Install Aide API

The Install Aide enables your desktop application or installer to install Palm OS® databases (including applications) into main memory on a handheld or to cards present in the handheld's expansion slots. To perform these and other operations, the Install Aide does the following:

- Selects which install conduit to run based on the extension of the file you want to install. For example, .prc, .pdb, .pnc extensions are usually registered by the default install conduit, which installs these as databases in the handheld's main memory, whereas another install conduit may be registered to install any file type (*.*) to an expansion card.
- Copies the file to the correct directory on the desktop computer.
- Sets a flag that informs the HotSync® Manager to run the correct install conduit (the one registered to handle files of the type to be installed) on the next synchronization.

The next time the user synchronizes, HotSync Manager runs the appropriate install conduits, which transfer files between the handheld and desktop.

The Install Aide functions are available in `InstAide.dll` and declared in `InstAide.h`. For more information on the Install Aide, see “[Using the Install Aide API](#)” on page 128 in the *C/C++ Sync Suite Companion*.

The sections in this chapter are:

Install Aide API Versions	952
Install Aide Structures	953
Install Aide Constants	955
Install Aide Functions	958
Install Aide Error Codes	1004

Install Aide API

Install Aide API Versions

Install Aide API Versions

The Install Aide API continues to evolve with new functions and new versions of existing functions. Beginning with version 4.0, each version of the Install Aide API has a major version number and a minor version number. You can determine the version of the Install Aide API that you are using by calling the [PlmGetLibVersion\(\)](#) function, available in Install Aide API versions 4.0 and later. Earlier versions do not have the `PlmGetLibVersion()` function—which indicates that version 4.0 and later functionality is not present.

The Install Aide API maintains backward compatibility within a major version. The minor version number changes when new functions are added or bugs are fixed. This document includes version information for each function.

If your application depends on functions that are available only in certain versions of the Install Aide API, you need to determine the version of the Install Aide API with which you are dealing on a specific installation. To do so, call the `PlmGetLibVersion()` function, which returns both the major version number and minor version number of the Install Aide API on the desktop computer.

For example, if your application depends on a function that was added in version 4.0 of the Install Aide API, you need to call `PlmGetLibVersion()` and then verify that the major number is 4 or greater and that the minor version number is 0 or greater.

For a summary of new features and other changes in the Install Aide API, see [Appendix A, “Revision History,”](#) on page 1105.

Install Aide Structures

This section describes the [FileInstallType](#) data structure used with the Install Aide API.

FileInstallType Struct

Purpose Receives information about an install conduit from. This information is stored in the conduit configuration entries and retrieved by [PltGetInstallConduitInfo\(\)](#) and [PltGetInstallCreatorInfo\(\)](#).

Declared In InstAide.h

Prototype

```
typedef struct {
    TCHAR szDir[ 64 ];
    TCHAR szExt[ 256 ];
    DWORD dwMask;
    TCHAR szModule[ 256 ];
    DWORD dwCreatorID;
    TCHAR szName[ 256 ];
} FileInstallType
```

Fields szDir

The value of the [Directory](#) entry, which defines the name of the handheld-install directory associated with this install conduit—for example, “Install”.

szExt

The value of the [Extensions](#) entry, which defines the file type extensions supported by this install conduit.

dwMask

The value of the [Mask](#) entry, which is a unique bit mask associated with this install conduit.

szModule

The value of the [Module](#) entry, which defines the filename of this install conduit DLL—for example, “inscn20.dll”.

dwCreatorID

The value of the [CreatorID](#) entry, which is the unique ID associated with this install conduit.

Install Aide API

FileInstallType

szName

The value of the [Name](#) entry, which specifies the name of the conduit—for example, “Install Conduit”.

Comments	For details on configuration entries that correspond to the fields of this structure, see “ Install Conduit Configuration Entries ” on page 184.
Compatibility	Install Aide version: All. Palm OS version: All.
See Also	PltGetInstallConduitInfo() , PltGetInstallCreatorInfo()

Install Aide Constants

This section describes the following groups of preprocessor constants that you use with the Install Aide API.

<u>Path Definitions</u>	Define the paths to get or set with <u>PltSetPath()</u> and <u>PltGetInstallFileFilter()</u> .
<u>Structure Sizes</u>	Define the sizes of structures used by the Install Aide.
<u>Version of the Install Aide API</u>	Defines the Install Aide API version that <u>P1mGetLibVersion()</u> passes back.

Path Definitions

Purpose Define the paths to get or set with [PltSetPath\(\)](#) and [PltGetInstallFileFilter\(\)](#).

Declared In InstAide.h

Constants

```
#define PILOT_PATH_HOME 1
The path to the user directories on the desktop computer—
for example, C:\Documents and
Settings\<WinUsername>\My Documents\Palm OS
Desktop. This corresponds to the Core\Path configuration
entry.
```

```
#define PILOT_PATH_HOTSYNC 2
The full path and filename of the HotSync Manager
executable—for example, C:\Program
Files\PalmSource\Desktop\hotsync.exe. This
corresponds to the Core\HotSyncPath configuration entry.
```

```
#define PILOT_PATH_TUTORIAL 3
This constant is not used by the Install Aide.
```

```
#define PILOT_PATH_MAX PILOT_PATH_TUTORIAL
The maximum value that the sPathType parameter can
have in PltSetPath() and
PltGetInstallFileFilter().
```

Compatibility Install Aide version: All.
Palm OS versions: All.

See Also [PltSetPath\(\)](#), [PltGetInstallFileFilter\(\)](#)

Structure Sizes

Purpose	Define the sizes of structures used by the Install Aide.
Declared In	<code>InstAide.h</code>
Constants	<code>#define FILEINSTALLTYPE_SIZE sizeof(FileInstallType)</code> Defines the size of the FileInstallType structure.
Compatibility	Install Aide version: All. Palm OS versions. All.
See Also	FileInstallType

Version of the Install Aide API

Purpose	Defines the Install Aide API version that PlmGetLibVersion() passes back.
Declared In	<code>InstAide.h</code>
Constants	<code>#define INSTAIDE_LIB_VER_MAJOR 4</code> Indicates the major version number. <code>#define INSTAIDE_LIB_VER_MINOR 1</code> Indicates the minor version number.
Comments	Versions 4.1 and later of the Install Aide API define these values, which <code>PlmGetLibVersion()</code> passes back. In version 4.0, the first version in which <code>PlmGetLibVersion()</code> is available, these constants were not defined, though <code>PlmGetLibVersion()</code> returned 4 for the major version number and 0 for the minor version number.
Compatibility	Install Aide version: 4.1 or later. Palm OS version: All.
See Also	PlmGetLibVersion()

Install Aide API

Install Aide Functions

Install Aide Functions

This section describes the following functions, which an application or conduit can use to install files or databases on a handheld or expansion card.

PlmGetLibVersion()	Retrieves the version of the Install Aide API.
PlmGetUserIDFromName()	Retrieves the DWORD user ID for the specified string user name.
PlmGetUserNameFromID()	Retrieves the string user name for the specified DWORD user ID.
PlmMoveInstallFileToHandheld()	Changes the destination of a file that is already queued to be installed in secondary storage in an expansion <i>slot</i> instead to be installed in primary storage on a user's <i>handheld</i> .
PlmMoveInstallFileToSlot()	Changes the destination of a file that is already queued to be installed in primary storage on a user's <i>handheld</i> instead to be installed in secondary storage in an expansion <i>slot</i> .
PlmSlotGetFileCount()	Retrieves the number of files queued to install to the specified slot for a given user.
PlmSlotGetFileInfo()	Returns the filename and file size of the specified file in the specified slot-install directory for a given user.
PlmSlotInstallFile()	Queues a file to be installed in secondary storage in an expansion slot of a user's handheld.
PlmSlotMoveInstallFile()	Changes from one expansion <i>slot</i> to another the destination of a file that is already queued to be installed in secondary storage in an expansion <i>slot</i> of a user's handheld.

[PlmSlotRemoveInstallFile\(\)](#)

Removes a file from the queue of files that are to be installed in secondary storage in an expansion *slot* of a user's handheld.

[PltGetFileCount\(\)](#)

Determines how many files of the specified type are contained in the specified user's handheld-install directory.

[PltGetFileInfo\(\)](#)

Returns information about a file in the specified handheld-install directory of a given user.

[PltGetFileName\(\)](#)

Retrieves the name of a file in the specified handheld-install directory of a given user, based on the file type and index of the file.

[PltGetFileTypeExtension\(\)](#)

Retrieves a supported file type extension.

[PltGetFileTypesCount\(\)](#)

Returns the number of file types registered with HotSync Manager for all install conduits.

[PltGetInstallConduitCount\(\)](#)

Returns the number of install conduits registered with HotSync Manager.

[PltGetInstallConduitInfo\(\)](#)

Retrieves information about the install conduit specified by an index.

[PltGetInstallCreatorInfo\(\)](#)

Retrieves information about the install conduit specified by unique ID.

[PltGetInstallFileFilter\(\)](#)

Retrieves all of the file filters for all install conduits on the desktop computer and concatenates the filter strings into one returned string.

[PltGetInstallFileFilterForUser\(\)](#)

Retrieves all of the file filters for all install conduits for the specified user on the desktop computer and concatenates the filter strings into one returned string.

[PltGetPath\(\)](#)

Retrieves the value of one of the defined Palm OS® Desktop path variables.

Install Aide API

Install Aide Functions

PltGetRegistryPath()	Retrieves the path to the HotSync configuration entries.
Plt GetUser()	Retrieves the name of the <i>i</i> th user stored in the users data file.
Plt GetUserCount()	Returns the number of users defined in the users data file on the desktop computer.
Plt GetUserDirectory()	Retrieves the full path to the specified user directory.
Plt InstallFile()	Queues a file to be installed in primary storage on a user's handheld.
Plt IsInstallMaskSet()	Determines whether the specified install conduit is set to run for the specified user during the next HotSync operation.
Plt IsUserProfile()	Determines whether an account is a user profile.
Plt RemoveInstallFile()	Removes a file from the queue of files that are to be installed in primary storage on a user's <i>handheld</i> .
Plt RemoveInstallRegistry()	Removes the install conduit settings that are set by Plt SetInstallRegistry() .
Plt ResetInstallMask()	Deselects the specified install conduit to run for the specified user during the next HotSync operation.
Plt SetInstallRegistry()	Restores install conduit information to default values.
Plt SetPath()	Sets the value of one of the Palm OS® Desktop path variables.

PlmGetLibVersion Function

Purpose	Retrieves the version of the Install Aide API.
Declared In	<code>InstAide.h</code>
Prototype	<code>int PlmGetLibVersion (DWORD *pdwMajor, DWORD *pdwMinor)</code>
Parameters	<p>$\leftarrow pdwMajor$ The major version number, which is defined by <code>INSTAIDE_LIB_VER_MAJOR</code>.</p> <p>$\leftarrow pdwMinor$ The minor version number, which is defined by <code>INSTAIDE_LIB_VER_MINOR</code>.</p>
Returns	Always returns 0.
Comments	Call <code>PlmGetLibVersion()</code> to determine the version of the Install Aide API that you are using before calling any of its functions. For more information, see “ Install Aide API Versions ” on page 952. For details about what changes between versions, see Appendix A, “Revision History,” on page 1105. The version number constants that this function passes back are defined in “ Version of the Install Aide API ” on page 957.
Compatibility	Install Aide version: 4.0 or later. Palm OS version: All.

Install Aide API

PlmGetUserIDFromName

PlmGetUserIDFromName Function

Purpose Retrieves the DWORD user ID for the specified string user name.

Declared In InstAide.h

Prototype int PlmGetUserIDFromName (TCHAR **pUser*,
 DWORD **pdwID*)

Parameters → *pUser*
 The user name string.

← *pdwID*
 Upon return, the user ID.

Returns If successful, returns 0.

If unsuccessful, returns a negative error code value.

For descriptions of all Install Aide API error codes, see “[Install Aide Error Codes](#)” on page 1004.

Comments The user ID is used internally for a specified user name.

NOTE: To access information about users on the desktop, PalmSource, Inc. recommends that you use the [User Data API](#) instead.

Compatibility Install Aide version: All.
Palm OS version: All.

See Also [UmGetUserID\(\)](#)

PlmGetUserNameFromID Function

Purpose	Retrieves the string user name for the specified DWORD user ID.
Declared In	<code>InstAide.h</code>
Prototype	<pre>int PlmGetUserNameFromID (DWORD dwID, TCHAR *pUserBuffer, short *psUserBufSize)</pre>
Parameters	<p>→ <i>dwID</i> The user ID.</p> <p>← <i>pUserBuffer</i> A pointer to a character buffer. Upon return, this contains the user name.</p> <p>↔ <i>psUserBufSize</i> A pointer to a short value that specifies the size, in TCHARS, of the character buffer referenced by the <i>pUserBuffer</i> parameter. Upon return, this is the actual number of TCHARS that were stored into the buffer referenced by the <i>pUserBuffer</i> parameter.</p>
Returns	<p>If successful, returns an integer value that specifies the number of TCHARS actually written into the buffer referred to by <i>pUserBuffer</i>.</p> <p>If unsuccessful, returns a negative error code value.</p> <p>For descriptions of all Install Aide API error codes, see “Install Aide Error Codes” on page 1004.</p>
Comments	The user ID is used internally for a specified user name.
<hr/> <p>NOTE: To access information about users on the desktop, PalmSource, Inc. recommends that you use the User Data API instead.</p> <hr/>	
Compatibility	Install Aide version: All. Palm OS version: All.
See Also	UmGetUserName ()

Install Aide API

PlmMoveInstallFileToHandheld

PlmMoveInstallFileToHandheld Function

Purpose	Changes the destination of a file that is already queued to be installed in secondary storage in an expansion <i>slot</i> instead to be installed in primary storage on a user's <i>handheld</i> .
Declared In	<code>InstAide.h</code>
Prototype	<code>int PlmMoveInstallFileToHandheld (DWORD dwUserId, DWORD dwSlotIdFrom, const TCHAR *pszFileName)</code>
Parameters	<p>→ <i>dwUserId</i> The ID of the user.</p> <p>→ <i>dwSlotIdFrom</i> The ID of the <i>source</i> slot (the one <i>from</i> which to move the file). The file currently exists in the slot-install directory of this slot.</p> <p>→ <i>pszFileName</i> A pointer to a character buffer that contains only the filename (no path) of the file to be moved.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following negative error code values:</p> <p><code>ERR_PILOT_INVALID_FILENAME</code> <code>ERR_PILOT_INVALID_PATH</code> <code>ERR_PALM_FILE_MOVE_FAILED</code></p> <p>For descriptions of all Install Aide API error codes, see “Install Aide Error Codes” on page 1004.</p>

Comments `PlmMoveInstallFileToHandheld()` moves a file on the desktop computer from the specified slot-install directory to the associated handheld-install directory—that is, to the directory associated with the install conduit registered to handle files of the specified type. This function accepts only file types that are registered with an install conduit—for example, .prc, .pdb, and .pqa file types are registered with the Install conduit that ships with HotSync Manager, so this function can move such files to a user's handheld-install directory.

To get slot IDs, use the User Data API's [UmSlotGetInfo\(\)](#) function.

Example `int retval = PlmMoveInstallFileToHandheld (dwUserId,
dwSlotIdFrom, "SummitSch.pdb");`

Compatibility Install Aide version: 4.0 or later.
Palm OS version: 4.0 or later.

See Also [UmSlotGetInfo\(\)](#), [PlmMoveInstallFileToSlot\(\)](#),
[PlmSlotMoveInstallFile\(\)](#)

Install Aide API

PlmMoveInstallFileToSlot

PlmMoveInstallFileToSlot Function

Purpose	Changes the destination of a file that is already queued to be installed in primary storage on a user's <i>handheld</i> instead to be installed in secondary storage in an expansion <i>slot</i> .
Declared In	<code>InstAide.h</code>
Prototype	<code>int PlmMoveInstallFileToSlot (DWORD dwUserId, const TCHAR *pszFileName, DWORD dwSlotIdTo)</code>
Parameters	<p>→ <i>dwUserId</i> The ID of the user.</p> <p>→ <i>pszFileName</i> A pointer to a character buffer that contains only the filename (no path) of the file to be moved.</p> <p>→ <i>dwSlotIdTo</i> The ID of the <i>destination</i> slot (the one <i>to</i> which to move the file). The file is to be moved to the install directory of this slot.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following negative error code values: <code>ERR_PILOT_INVALID_FILENAME</code> <code>ERR_PILOT_INVALID_PATH</code> <code>ERR_PALM_FILE_MOVE_FAILED</code> For descriptions of all Install Aide API error codes, see “ Install Aide Error Codes ” on page 1004.

Comments PlmMoveInstallFileToSlot() moves a file on the desktop computer from the associated handheld-install directory—that is, the directory associated with the install conduit registered to handle files of the type to be moved—to the specified slot-install directory. This function accepts only file types that are registered with an install conduit—for example, .prc, .pdb, and .pqa file types are registered with the Install conduit that ships with HotSync Manager, so this function can move such files to a user's slot-install directory.

To get slot IDs, use the User Data API's [UmSlotGetInfo\(\)](#) function.

Example
int retval = PlmMoveInstallFileToSlot (dwUserId,
 "SummitSch.pdb", dwSlotIdTo);

Compatibility Install Aide version: 4.0 or later.
Palm OS version: 4.0 or later.

See Also [UmSlotGetInfo\(\)](#), [PlmMoveInstallFileToHandheld\(\)](#),
[PlmSlotMoveInstallFile\(\)](#)

Install Aide API

PlmSlotGetFileCount

PlmSlotGetFileCount Function

Purpose Retrieves the number of files queued to install to the specified slot for a given user.

Declared In InstAide.h

Prototype int PlmSlotGetFileCount (DWORD *dwUserId*,
 DWORD *dwSlotId*, long **plFileCount*)

Parameters → *dwUserId*
The ID of the user.

→ *dwSlotId*
The ID of the slot for which to return the file count.

← *plFileCount*
A pointer to the number of files.

Returns If successful, returns 0.

If unsuccessful, returns one of the following negative error code values:

ERR_PILOT_INVALID_PATH

ERR_PILOT_INVALID_BUFFER

For descriptions of all Install Aide API error codes, see “[Install Aide Error Codes](#)” on page 1004.

Comments PlmSlotGetFileCount () determines the number of files of all types that exist in the specified slot-install directory for a specific user. This directory holds all the files that will be installed on the specified expansion slot on the handheld during the next HotSync operation.

To get slot IDs, use the User Data API’s [UmSlotGetInfo\(\)](#) function.

Example

```
long nFileCount;
int retval = PlmSlotGetFileCount (dwUserId, dwSlotId,
&nFileCount);
```

Compatibility Install Aide version: 4.0 or later.
Palm OS version: 4.0 or later.

See Also [PltGetFileCount\(\)](#)

PlmSlotGetFileInfo Function

Purpose Returns the filename and file size of the specified file in the specified slot-install directory for a given user.

Declared In InstAide.h

Prototype

```
int PlmSlotGetFileInfo (DWORD dwUserId,  
                      DWORD dwSlotId, unsigned int iIndex,  
                      TCHAR *pszFileName, long *plFileSize,  
                      DWORD *pdwFileSize)
```

Parameters

→ *dwUserId*
The ID of the user.

→ *dwSlotId*
The ID of the slot from which to get the file's information.

→ *iIndex*
A zero-based file index. This index specifies which file to access in the slot-install directory.

← *pszFileName*
A pointer to a character buffer. Upon return, this buffer holds the filename of the file that is referenced by *iIndex*.

↔ *plFileSize*
Upon entry, a pointer to a long value that specifies the size, in bytes, of the *pszFileName* parameter. Upon return, this is the filename length (the actual number of TCHARs that were stored into the buffer referenced by the *pszFileName* parameter).

← *pdwFileSize*
Upon return, a pointer to a DWORD that is the size of the file. The size (in bytes) of files transferred to the card are assumed not to exceed a DWORD.

Returns

If successful, returns the size of the string copied into the *pszFileName* buffer.

If unsuccessful, returns one of the following negative error code values:

ERR_PILOT_BUFSIZE_TOO_SMALL
ERR_PILOT_INVALID_PATH
ERR_PILOT_INVALID_BUFFER

Install Aide API

PlmSlotGetFileInfo

For descriptions of all Install Aide API error codes, see “[Install Aide Error Codes](#)” on page 1004.

Comments To use `PlmSlotGetFileInfo()`, specify the user, the slot ID, and the file’s index in the slot-install directory.

`PlmSlotGetFileInfo()` retrieves the filename and file size. This function works with all file types.

To get slot IDs, use the User Data API’s [`UmSlotGetInfo\(\)`](#) function.

If the `pszFileName` buffer is too small to contain the filename, this function returns the error `ERR_PILOT_BUFSIZE_TOO_SMALL` and stores the actual required size into the `plFileBufSize` parameter. `ERR_PILOT_INVALID_PATH` is returned if there was a problem retrieving the path of the user’s slot-install directory.

Example Typically you call [`PlmSlotGetFileCount\(\)`](#) and then call `PlmSlotGetFileInfo()` in a loop to get the filename and size for each file, as shown below.

```
long nFileCount;
    int retval = PlmSlotGetFileCount (dwUserId, dwSlotId,
        &nFileCount);
    TCHAR szFileName [256];
    for (int i = 0; i < nFileCount; i++)
    {
        dwFileSize = sizeof (szFileName);
        PlmSlotGetFileInfo (dwUserId, dwSlotId, i, szFileName,
            &nBufSize, &dwFileSize);
    }
```

Compatibility Install Aide version: 4.0 or later.

Palm OS version: 4.0 or later.

See Also [`PltGetFileInfo\(\)`](#), [`UmSlotGetInfo\(\)`](#)

PlmSlotInstallFile Function

Purpose	Queues a file to be installed in secondary storage in an expansion slot of a user's handheld.
Declared In	<code>InstAide.h</code>
Prototype	<pre>int PlmSlotInstallFile (DWORD dwUserId, DWORD dwSlotId, const TCHAR *pszFileName)</pre>
Parameters	<p>→ <i>dwUserId</i> The ID of the user.</p> <p>→ <i>dwSlotId</i> The ID of the slot to which to queue the file for installation.</p> <p>→ <i>pszFileName</i> A pointer to a character buffer that contains the full path and filename of the file to install.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following negative error code value:</p> <p><code>ERR_PILOT_INVALID_FILENAME</code> <code>ERR_PILOT_INVALID_FILE_TYPE</code> <code>ERR_PILOT_INVALID_PATH</code> <code>ERR_PILOT_FILE_ALREADY_EXISTS</code> <code>ERR_PILOT_INVALID_SOURCE_FILE</code> <code>ERR_PILOT_COPY_FAILED</code></p> <p>For descriptions of all Install Aide API error codes, see “Install Aide Error Codes” on page 1004.</p>

Install Aide API

PlmSlotInstallFile

Comments `PlmSlotInstallFile()` copies the file specified by `pszFileName` into the slot-install directory specified by `dwUserId` and `dwSlotId`, and then modifies the configuration entries to notify HotSync Manager that it needs to install the file during the next HotSync operation. This function accepts all file types.

To get slot IDs, use the User Data API's [UmSlotGetInfo\(\)](#) function.

Example

```
int retval = PlmSlotInstallFile (dwUserId, dwSlotId,
    "C:\MyDocuments\SummitAgenda.doc");
```

Compatibility Install Aide version: 4.0 or later.
Palm OS version: 4.0 or later.

See Also [PltInstallFile\(\)](#), [PlmSlotRemoveInstallFile\(\)](#),
[UmSlotGetInfo\(\)](#)

PlmSlotMoveInstallFile Function

Purpose Changes from one expansion *slot* to another the destination of a file that is already queued to be installed in secondary storage in an expansion *slot* of a user's handheld.

Declared In InstAide.h

Prototype

```
int PlmSlotMoveInstallFile (DWORD dwUserId,  
                           DWORD dwSlotIdFrom, const TCHAR *pszFileName,  
                           DWORD dwSlotIdTo)
```

Parameters

- *dwUserId*
The ID of the user.
- *dwSlotIdFrom*
The ID of the *source* slot (the one *from* which to move the file).
The file currently exists in the install directory of this slot.
- *pszFileName*
A pointer to a character buffer that contains only the filename (no path) of the file to be moved.
- *dwSlotIdTo*
The ID of the *destination* slot (the one *to* which to move the file). The file is to be moved to the install directory of this slot.

Returns If successful, returns 0.

If unsuccessful, returns one of the following negative error code values:

ERR_PILOT_INVALID_FILENAME

ERR_PILOT_INVALID_PATH

ERR_PALM_FILE_MOVE_FAILED

For descriptions of all Install Aide API error codes, see “[Install Aide Error Codes](#)” on page 1004.

Install Aide API

PlmSlotMoveInstallFile

Comments `PlmSlotMoveInstallFile()` moves a file on the desktop computer from one slot-install directory to another slot-install directory. This function is useful for users whose handhelds have multiple slots. This function accepts all file types.

To get slot IDs, use the User Data API's [UmSlotGetInfo\(\)](#) function.

Example

```
int retval = PlmSlotMoveInstallFile (dwUserId, dwSlotIdFrom,
"SummitAgenda.doc", dwSlotIdTo);
```

Compatibility Install Aide version: 4.0 or later.
Palm OS version: 4.0 or later.

See Also [PlmMoveInstallFileToHandheld\(\)](#),
[PlmMoveInstallFileToSlot\(\)](#), [UmSlotGetInfo\(\)](#)

PlmSlotRemoveInstallFile Function

Purpose Removes a file from the queue of files that are to be installed in secondary storage in an expansion *slot* of a user's handheld.

Declared In InstAide.h

Prototype

```
int PlmSlotRemoveInstallFile (DWORD dwUserId,  
    DWORD dwSlotId, const TCHAR *pszFileName)
```

Parameters

→ *dwUserId*
The ID of the user.

→ *dwSlotId*
The ID of the slot from which to remove the file.

→ *pszFileName*
A pointer to a character buffer that contains the filename of the file to be removed; no path is required.

Returns If successful, returns 0.

If unsuccessful, returns one of the following negative error code values:

ERR_PILOT_INVALID_FILENAME

ERR_PILOT_INVALID_PATH

ERR_PALM_FILE_DELETE_FAILED

For descriptions of all Install Aide API error codes, see “[Install Aide Error Codes](#)” on page 1004.

Comments PlmSlotRemoveInstallFile() removes a file on the desktop computer that was queued in a slot-install directory for a given user. This function accepts all file types.

To get slot IDs, use the User Data API’s [UmSlotGetInfo\(\)](#) function.

Example

```
int retval = PlmSlotRemoveInstallFile (dwUserId, dwSlotId,  
    "SummitAgenda.doc");
```

Compatibility Install Aide version: 4.0 or later.
Palm OS version: 4.0 or later.

See Also [PltRemoveInstallFile\(\)](#), [UmSlotGetInfo\(\)](#)

PltGetFileCount Function

Purpose	Determines how many files of the specified type are contained in the specified user's handheld-install directory.
Declared In	InstAide.h
Prototype	<pre>int PltGetFileCount (TCHAR *pUser, TCHAR *pExtension)</pre>
Parameters	<p>→ <i>pUser</i> The name of the user.</p> <p>→ <i>pExtension</i> The file type extension. See “Specifying File Types” on page 133 in the <i>C/C++ Sync Suite Companion</i>.</p>
Returns	If successful, returns the number of files of the specified type in the user's handheld-install directory. If unsuccessful, returns a negative error code value. For descriptions of all Install Aide API error codes, see “ Install Aide Error Codes ” on page 1004.
Comments	PltGetFileCount () determines the number of files of a specific type that exist in the user's handheld-install directory that is associated with the install conduit registered to handle files with the specified extension. This count does not include files in the slot-install directories. To count files in slot-install directories, call the PlmSlotGetFileCount () function.
Compatibility	Install Aide version: All. Palm OS version: All.
See Also	PlmSlotGetFileCount ()

PltGetFileInfo Function

Purpose Returns information about a file in the specified handheld-install directory of a given user.

Declared In InstAide.h

Prototype

```
int PltGetFileInfo (TCHAR *pUser,
                    unsigned int iIndex, TCHAR *pExtension,
                    TCHAR *pFileName, short *psFileSize,
                    DWORD *pdwFileSize)
```

Parameters

→ *pUser*
The user name.

→ *iIndex*
The index of the file in the handheld-install directory. This is a zero-based index.

→ *pExtension*
The file type extension. See “[Specifying File Types](#)” on page 133 in the *C/C++ Sync Suite Companion*.

← *pFileName*
Upon return, the name of the file.

↔ *psFileSize*
A pointer to a short value that specifies the size, in TCHARS, of the character buffer referenced by the *pFileName* parameter. Upon return, this is the actual number of TCHARS that were stored into the buffer referenced by the *pFileName* parameter.

← *pdwFileSize*
Upon return, the size of the file in bytes.

Returns

If successful, returns the size of the string copied into the *pFileName* buffer.

If unsuccessful, returns a negative error code value.

For descriptions of all Install Aide API error codes, see “[Install Aide Error Codes](#)” on page 1004.

Install Aide API

PltGetFileInfo

Comments	To use <code>PltGetFileInfo()</code> , you need to specify the file's type and its index in the associated handheld-install directory. <code>PltGetFileInfo()</code> retrieves the filename and file size. If the size of the buffer that you supply is too small to contain the filename, this function returns the error <code>ERR_PILOT_BUFSIZE_TOO_SMALL</code> and stores the actual required size into the <code>psFileBufSize</code> parameter. This function is similar to the PltGetFileName() function, except that it returns the file size in addition to the filename.
Compatibility	Install Aide version: All. Palm OS version: All.
See Also	PltGetFileName()

PltGetFileName Function

Purpose Retrieves the name of a file in the specified handheld-install directory of a given user, based on the file type and index of the file.

Declared In InstAide.h

Prototype

```
int PltGetFileName (TCHAR *pUser,
                    unsigned int iIndex, TCHAR *pExtension,
                    TCHAR *pFileName, short *psFileSize)
```

Parameters

→ *pUser*
The user name.

→ *iIndex*
The index of the file in the user's handheld-install directory.
This is a zero-based index.

→ *pExtension*
The file type extension. See “[Specifying File Types](#)” on page 133 in the *C/C++ Sync Suite Companion*.

← *pFileName*
Upon return, the name of the file.

↔ *psFileSize*
A pointer to a short value that specifies the size, in TCHARS, of the character buffer referenced by the *pFileName* parameter. Upon return, this is the actual number of TCHARS that were stored into the buffer referenced by the *pFileName* parameter.

Returns

If successful, returns the size of the string copied into the *pFileName* buffer.

If unsuccessful, returns a negative error code value.

For descriptions of all Install Aide API error codes, see “[Install Aide Error Codes](#)” on page 1004.

Install Aide API

PltGetFileName

Comments	PltGetFileName() iterates through the files of the specified type in the user's associated handheld-install directory. Use an index value of 0 to retrieve the name of the first file, and increment the index by 1 for each subsequent file of the specified type. If the size of the buffer that you supply is too small to contain the filename, this function returns the error ERR_PILOT_BUFSIZE_TOO_SMALL and stores the actual required size into the <i>psFileBufSize</i> parameter.
Compatibility	Install Aide version: All. Palm OS version: All.
See Also	PltGetFileInfo() , PltGetFileTypeExtension()

PltGetFileTypeExtension Function

Purpose	Retrieves a supported file type extension.
Declared In	InstAide.h
Prototype	<pre>int PltGetFileTypeExtension (int iIndex, TCHAR *pBuffer, int *piBufferSize)</pre>
Parameters	<p>→ <i>iIndex</i> An index into the list of filename extensions.</p> <p>← <i>pBuffer</i> Upon return, the extension string. See “Specifying File Types” on page 133 in the <i>C/C++ Sync Suite Companion</i>.</p> <p>↔ <i>piBufferSize</i> Upon entry, a pointer to an integer value that specifies the size, in TCHARs, of the character buffer referenced by the <i>pBuffer</i> parameter. Upon return, this is the actual number of TCHARs that were stored into the buffer referenced by the <i>pBuffer</i> parameter.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns a negative error code value.</p> <p>For descriptions of all Install Aide API error codes, see “Install Aide Error Codes” on page 1004.</p>
Comments	<p><code>PltGetFileTypeExtension()</code> iterates through the file types that your application supports. Use an index value of 0 to retrieve the first supported file type extension, and increment the index value by 1 for each subsequent file type.</p> <p>After retrieving an extension string, you can call PltGetFileName() or PltGetFileInfo() to iterate through the files of that type in the user’s associated handheld-install directory.</p> <p>If the size of the buffer that you supply is too small to contain the extension string, this function returns the error <code>ERR_PILOT_BUFSIZE_TOO_SMALL</code> and stores the actual required size into the <i>piBufferSize</i> parameter.</p>

Install Aide API

PltGetFileTypeExtension

Compatibility Install Aide version: All.
Palm OS version: All.

See Also [PltGetFileCount\(\)](#), [PltGetFileInfo\(\)](#),
[PltGetFileName\(\)](#)

PltGetFileTypesCount Function

Purpose	Returns the number of file types registered with HotSync Manager for all install conduits.
Declared In	<code>InstAide.h</code>
Prototype	<code>int PltGetFileTypesCount (void)</code>
Parameters	None.
Returns	If successful, returns the number of file types registered with HotSync Manager. If unsuccessful, returns a negative error code value. For descriptions of all Install Aide API error codes, see " Install Aide Error Codes " on page 1004.
Comments	<code>PltGetFileTypesCount()</code> determines the number of file types registered with HotSync Manager for all install conduits. You can then iterate through those types using <code>PltGetTypeExtension()</code> .
Compatibility	Install Aide version: All. Palm OS version: All.
See Also	<code>PltGetTypeExtension()</code>

Install Aide API

PltGetInstallConduitCount

PltGetInstallConduitCount Function

Purpose	Returns the number of install conduits registered with HotSync Manager.
Declared In	<code>InstAide.h</code>
Prototype	<code>int PltGetInstallConduitCount (void)</code>
Parameters	None.
Returns	If successful, returns the number of install conduits registered with HotSync Manager, as defined in the configuration entries. If unsuccessful, returns a negative error code value. For descriptions of all Install Aide API error codes, see “ Install Aide Error Codes ” on page 1004.
Comments	<code>PltGetInstallConduitCount()</code> determines the number of install conduits registered with HotSync Manager. You can then iterate through the install conduits with <code>PltGetInstallConduitInfo()</code> . Note that install conduits are different from other conduits (known as application or component conduits), which are not included in this count.
Compatibility	Install Aide version: All. Palm OS version: All.
See Also	PltGetInstallConduitInfo()

PltGetInstallConduitInfo Function

Purpose	Retrieves information about the install conduit specified by an index.
Declared In	<code>InstAide.h</code>
Prototype	<pre>int PltGetInstallConduitInfo (unsigned int <i>iIndex</i>, FileInstallType *<i>pBuf</i>, int *<i>piBufferSize</i>)</pre>
Parameters	<p>$\rightarrow iIndex$ An index into the list of install conduit information blocks in the configuration entries.</p> <p>$\leftarrow pBuf$ A pointer to a <code>FileInstallType</code> structure. Upon return, the structure is filled with information from the configuration entries about the specified install conduit.</p> <p>$\leftrightarrow piBufferSize$ A pointer to an integer value that specifies the size, in bytes, of the buffer referenced by the <i>pBuf</i> parameter. Upon return, this is the actual number of bytes that were stored into the buffer referenced by the <i>pBuf</i> parameter.</p>
Returns	<p>If successful, returns the constant <code>FILEINSTALLTYPE_SIZE</code> to indicate that it correctly stored information into the buffer.</p> <p>If unsuccessful, returns a negative error code value.</p> <p>For descriptions of all Install Aide API error codes, see “Install Aide Error Codes” on page 1004.</p>
Comments	<p><code>PltGetInstallConduitInfo()</code> retrieves information about the specified install conduit registered with HotSync Manager. Use an index value of 0 to retrieve information about the first install conduit, and increment the index value by 1 for each subsequent conduit.</p> <p>If the size of the buffer that you supply is too small to contain the structure, this function returns the error <code>ERR_PILOT_BUFSIZE_TOO_SMALL</code> and stores the actual required size into the <i>piBufferSize</i> parameter.</p>
Compatibility	Install Aide version: All. Palm OS version: All.
See Also	PltGetInstallConduitCount()

Install Aide API

PltGetInstallCreatorInfo

PltGetInstallCreatorInfo Function

Purpose	Retrieves information about the install conduit specified by unique ID.
Declared In	<code>InstAide.h</code>
Prototype	<code>int PltGetInstallCreatorInfo (DWORD dwCreatorID, FileInstallType *pBuf, int *piBufferSize)</code>
Parameters	<p>→ <i>dwCreatorID</i> The unique ID of the install conduit whose information you want.</p> <p>← <i>pBuf</i> A pointer to a FileInstallType structure. Upon return, the structure is filled with information from the configuration entries about the install conduit.</p> <p>↔ <i>piBufferSize</i> A pointer to an integer value that specifies the size, in bytes, of the buffer referenced by the <i>pBuf</i> parameter. Upon return, this is the actual number of bytes that were stored into the buffer referenced by the <i>pBuf</i> parameter.</p>
Returns	If successful, returns the constant <code>FILEINSTALLTYPE_SIZE</code> to indicate that it correctly stored information into the buffer. If unsuccessful, returns a negative error code value. For descriptions of all Install Aide API error codes, see “ Install Aide Error Codes ” on page 1004.
Comments	<code>PltGetInstallCreatorInfo()</code> retrieves information about an install conduit, given its unique ID. If the size of the buffer that you supply is too small to contain the structure, this function returns the error <code>ERR_PILOT_BUFSIZE_TOO_SMALL</code> and stores the actual required size into the <i>piBufferSize</i> parameter.
Compatibility	Install Aide version: All. Palm OS version: All.

PltGetInstallFileFilter Function

Purpose	Retrieves all of the file filters for all install conduits on the desktop computer and concatenates the filter strings into one returned string.
Declared In	<code>InstAide.h</code>
Prototype	<code>int PltGetInstallFileFilter (TCHAR *pFilter, int *piBufferSize)</code>
Parameters	<p>$\leftarrow pFilter$ A pointer to a character buffer. Upon return, the buffer contains a string that specifies the file filter types (see “Specifying File Types” on page 133 in the <i>C/C++ Sync Suite Companion</i>).</p> <p>$\leftrightarrow piBufferSize$ A pointer to an integer value that specifies the size, in TCHARs, of the buffer referenced by the <i>pFilter</i> parameter. Upon return, this is the actual number of TCHARs that were stored into the buffer referenced by the <i>pFilter</i> parameter.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns a negative error code value.</p> <p>For descriptions of all Install Aide API error codes, see “Install Aide Error Codes” on page 1004.</p>
Comments	<p><code>PltGetInstallFileFilter()</code> passes back a single string containing all the file type filters that are supported by all of the install conduits registered with HotSync Manager for all the HotSync users created by the current Windows user. To limit the scope to a specific HotSync user, call the <code>PltGetInstallFileFilterForUser()</code>.</p> <p>If the size of the buffer that you supply is too small to contain the filter string, this function returns the error <code>ERR_PILOT_BUFSIZE_TOO_SMALL</code> and stores the actual required size into the <i>piBufferSize</i> parameter.</p>
Compatibility	Install Aide version: All. Palm OS version: All.
See Also	<code>PltGetInstallFileFilterForUser()</code>

Install Aide API

PltGetInstallFileFilterForUser

PltGetInstallFileFilterForUser Function

Purpose Retrieves all of the file filters for all install conduits for the specified user on the desktop computer and concatenates the filter strings into one returned string.

Declared In InstAide.h

Prototype int PltGetInstallFileFilterForUser (TCHAR **pUser*,
 TCHAR **pFilter*, int **piBufferSize*)

Parameters → *pUser*
 The ID of the user. If this value is NULL, this function returns file filters for all users.

← *pFilter*
 A pointer to a character buffer. Upon return, the buffer contains a string that specifies the file filter types (see “[Specifying File Types](#)” on page 133 in the C/C++ Sync Suite Companion).

↔ *piBufferSize*
 A pointer to an integer value that specifies the size, in TCHARs, of the buffer referenced by the *pFilter* parameter. Upon return, this is the actual number of TCHARs that were stored into the buffer referenced by the *pFilter* parameter.

Returns If successful, returns 0.
If unsuccessful, returns a negative error code value.
For descriptions of all Install Aide API error codes, see “[Install Aide Error Codes](#)” on page 1004.

Comments PltGetInstallFileFilterForUser () passes back a single string containing all the file type filters that are supported by all of the install conduits registered with HotSync Manager for the specified user on the desktop computer. To get this information for all users, call the [PltGetInstallFileFilter\(\)](#) function.

The difference between these two functions is that PltGetInstallFileFilterForUser () takes into account whether the specified user’s handheld has been synchronized with an expansion card in its slot. If it has, then this function includes the file types registered by any install conduits that install files to expansion cards. For example, it includes the “*.*” file type filter registered by the Install to Card install conduit, if the specified

user's handheld has been synchronized with an expansion card in its slot.

If the size of the buffer that you supply is too small to contain the filter string, this function returns the error
ERR_PILOT_BUFSIZE_TOO_SMALL and stores the actual required size into the *piBufferSize* parameter.

Compatibility Install Aide version: 4.0 or later.
 Palm OS version: 4.0 or later.

See Also [PltGetInstallFileFilter\(\)](#)

Install Aide API

PltGetPath

PltGetPath Function

Purpose	Retrieves the value of one of the defined Palm OS® Desktop path variables.
Declared In	<code>InstAide.h</code>
Prototype	<code>int PltGetPath (unsigned short <i>sPathType</i>, TCHAR *<i>pPathBuffer</i>, short *<i>psPathBufSize</i>)</code>
Parameters	<p>→ <i>sPathType</i> The path to retrieve. Specify one of the defined paths described in “Path Definitions” on page 956.</p> <p>← <i>pPathBuffer</i> A pointer to a character buffer. Upon return, this contains the path string.</p> <p>↔ <i>psPathBufSize</i> A pointer to a short value that specifies the size, in TCHARS, of the buffer referenced by the <i>pPathBuffer</i> parameter. Upon return, this is the actual number of TCHARS that were stored in the buffer referenced by the <i>pPathBuffer</i> parameter.</p>
Returns	If successful, returns an integer value that specifies the number of TCHARS actually written into the buffer referred to by <i>pPathBuffer</i> . If unsuccessful, returns a negative error code value. For descriptions of all Install Aide API error codes, see “ Install Aide Error Codes ” on page 1004.
Comments	If the size of the buffer that you supply is too small to contain the path string, this function returns the error <code>ERR_PILOT_BUFSIZE_TOO_SMALL</code> and stores the actual required size into the <i>psFileBufSize</i> parameter.
Compatibility	Install Aide version: All. Palm OS version: All.
See Also	pltSetPath()

PltGetRegistryPath Function

Purpose	Retrieves the path to the HotSync configuration entries.
Declared In	InstAide.h
Prototype	<pre>int PltGetRegistryPath (unsigned int <i>iIndex</i>, TCHAR *<i>pBuffer</i>, int *<i>piBufferSize</i>)</pre>
Parameters	<p>→ <i>iIndex</i> An index into the list of installed conduit information blocks in the configuration entries.</p> <p>← <i>pBuffer</i> A pointer to a character buffer. Upon return, this is the path string.</p> <p>↔ <i>piBufferSize</i> A pointer to an integer value that specifies the size, in TCHARs, of the buffer referred to by the <i>pBuffer</i> parameter. Upon return, this is the actual number of TCHARs that were stored into the buffer referenced by the <i>pBuffer</i> parameter.</p>
Returns	If successful, returns an integer value that specifies the number of TCHARs actually written into the buffer referred to by <i>pBuffer</i> . If unsuccessful, returns a negative error code value. For descriptions of all Install Aide API error codes, see “ Install Aide Error Codes ” on page 1004.
Comments	<p><code>PltGetRegistryPath()</code> retrieves the path on the desktop computer of the HotSync configuration entries.</p> <p>If the size of the buffer that you supply is too small to contain the path string, this function returns the error <code>ERR_PILOT_BUFSIZE_TOO_SMALL</code> and stores the actual required size into the <i>piBufferSize</i> parameter.</p>
Compatibility	<p>Install Aide version: All. Palm OS version: All.</p>

IMPORTANT: To access information in the configuration entries, PalmSource, Inc. recommends that you not use `PltGetRegistryPath`. Instead use the [Conduit Manager API](#), [Install Conduit Manager API](#), [User Data API](#), or other [Install Aide Functions](#).

Plt GetUser Function

Purpose Retrieves the name of the *i*th user stored in the users data file.

Declared In InstAide.h

Prototype int Plt GetUser (unsigned int *iIndex*,
 TCHAR **pUserBuffer*, short **psUserBufSize*)

Parameters → *iIndex*
 An index into the list of users in the users data file.

← *pUserBuffer*
 A pointer to a character buffer. Upon return, this is the user
 name.

↔ *psUserBufSize*
 A pointer to a short value that specifies the size, in TCHARS,
 of the buffer referred to by the *pUserBuffer* parameter.
 Upon return, this is the actual number of TCHARS that were
 stored into the buffer referenced by the *pUserBuffer*
 parameter.

Returns If successful, returns an integer value that specifies the number of
TCHARS actually written into the buffer referred to by
pUserBuffer.

If unsuccessful, returns a negative error code value.

For descriptions of all Install Aide API error codes, see “[Install Aide Error Codes](#)” on page 1004.

Comments Use Plt GetUser () to iterate through the list of users in the users
data file on the desktop computer.

If the size of the buffer that you supply is too small to contain the
user name, this function returns the error
ERR_PILOT_BUFSIZE_TOO_SMALL and stores the actual required
size into the *psUserBufSize* parameter.

NOTE: To access information about users on the desktop,
PalmSource, Inc. recommends that you use the [User Data API](#)
instead.

Compatibility Install Aide version: All.
Palm OS version: All.

See Also [UmGetUserName\(\)](#)

Install Aide API

PltGetUserCount

PltGetUserCount Function

Purpose	Returns the number of users defined in the users data file on the desktop computer.
Declared In	<code>InstAide.h</code>
Prototype	<code>int PltGetUserCount (void)</code>
Parameters	None.
Returns	If successful, returns a positive integer value that specifies the number of users defined in the users data file. If unsuccessful, returns a negative error code value. For descriptions of all Install Aide API error codes, see “ Install Aide Error Codes ” on page 1004.
Comments	<code>PltGetUserCount ()</code> determines how many users are registered in the users data file on the desktop computer. You can then use the <code>Plt GetUser()</code> function to iterate through the users.
<hr/> NOTE: To access information about users on the desktop, PalmSource, Inc. recommends that you use the User Data API instead.	
Compatibility	Install Aide version: All. Palm OS version: All.
See Also	<code>Plt GetUser()</code> , <code>UmGetUserCount()</code>

Plt GetUserDirectory Function

Purpose Retrieves the full path to the specified user directory.

Declared In InstAide.h

Prototype int Plt GetUserDirectory (TCHAR **pUser*,
 TCHAR **pBuffer*, int **piBufferSize*)

Parameters → *pUser*
 The user name.

← *pBuffer*
 A pointer to a character buffer. Upon return, this is the path
 to the user directory.

↔ *piBufferSize*
 A pointer to an integer value that specifies the size, in
 TCHARs, of the buffer referred to by the *pBuffer* parameter.
 Upon return, this is the actual number of TCHARs that were
 stored into the buffer referenced by the *pBuffer* parameter.

Returns If successful, returns an integer value that specifies the number of
TCHARs actually written into the buffer referred to by *pBuffer*.
If unsuccessful, returns a negative error code value.

For descriptions of all Install Aide API error codes, see “[Install Aide Error Codes](#)” on page 1004.

Comments Plt GetUserDirectory () determines the path of a specified user
directory on the desktop computer.

If the size of the buffer that you supply is too small to contain the
path name, this function returns the error
ERR_PILOT_BUFSIZE_TOO_SMALL and stores the actual required
size into the *piBufferSize* parameter.

NOTE: To access information about users on the desktop,
PalmSource, Inc. recommends that you use the [User Data API](#)
instead.

Compatibility Install Aide version: All.
Palm OS version: All.

See Also [Um GetUserDirectory\(\)](#)

Install Aide API

PltInstallFile

PltInstallFile Function

Purpose	Queues a file to be installed in primary storage on a user's handheld.
Declared In	<code>InstAide.h</code>
Prototype	<code>int PltInstallFile (TCHAR *pUser, TCHAR *pFileSpec)</code>
Parameters	<p>→ <i>pUser</i> The user name.</p> <p>→ <i>pFileSpec</i> The path of the file to install.</p>
Returns	If successful, returns 0. If unsuccessful, returns a negative error code value. For descriptions of all Install Aide API error codes, see “ Install Aide Error Codes ” on page 1004.
Comments	<code>PltInstallFile()</code> copies the file specified by <i>pFileSpec</i> into the handheld-install directory for the user name specified by <i>pUser</i> , and then sets an “Install” configuration entry to specify which install conduit that HotSync Manager must run during the next synchronization operation to install the specified file.
Compatibility	Install Aide version: All. Palm OS version: All.
See Also	PlmSlotInstallFile() , PltGetInstallConduitInfo()

PltIsInstallMaskSet Function

Purpose	Determines whether the specified install conduit is set to run for the specified user during the next HotSync operation.
Declared In	<code>InstAide.h</code>
Prototype	<pre>int PltIsInstallMaskSet (TCHAR *pUser, DWORD dwMask)</pre>
Parameters	<p>→ <i>pUser</i> The user name.</p> <p>→ <i>dwMask</i> A one-bit mask associated with the install conduit that you want to know the run status of.</p>
Returns	<p>If the specified install conduit is set to run, returns a value greater than 0.</p> <p>If the specified install conduit is set <i>not</i> to run, returns a value of 0.</p> <p>If unsuccessful, returns a negative error code value.</p> <p>For descriptions of all Install Aide API error codes, see “Install Aide Error Codes” on page 1004.</p>
Comments	During a HotSync operation, HotSync Manager ANDs the <i>dwMask</i> bit mask value you provide (corresponding to the install conduit you wish to check the run status of) with the “Install” configuration entry for the specified user. This function returns a value >0 if the bit mask value you specified is set; otherwise it returns a value of 0. If you do not know the <i>dwMask</i> value, use <code>PltGetInstallConduitInfo()</code> to get the <i>dwMask</i> value for each install conduit.
Compatibility	Install Aide version: All. Palm OS version: All.
See Also	<code>PltGetInstallConduitInfo()</code>

Install Aide API

PltIsUserProfile

PltIsUserProfile Function

Purpose	Determines whether an account is a user profile.
Declared In	InstAide.h
Prototype	int PltIsUserProfile (TCHAR * <i>pUser</i>)
Parameters	$\rightarrow pUser$ The user name.
Returns	If this account is a user profile, returns 1. If this account is a regular user account, returns 0. If unsuccessful, returns a negative error code value. For descriptions of all Install Aide API error codes, see “ Install Aide Error Codes ” on page 1004.
Comments	For more information on user profiles , see the glossary in the <i>Introduction to Conduit Development</i> .
<hr/> NOTE: To access information about users on the desktop, PalmSource, Inc. recommends that you use the User Data API instead.	
Compatibility	Install Aide version: All. Palm OS version: All.
See Also	UmIsUserProfile()

PltRemoveInstallFile Function

Purpose	Removes a file from the queue of files that are to be installed in primary storage on a user's <i>handheld</i> .
Declared In	InstAide.h
Prototype	<pre>int PltRemoveInstallFile (TCHAR *pUser, TCHAR *pFileName)</pre>
Parameters	<p>→ <i>pUser</i> The user's name.</p> <p>→ <i>pFileName</i> The name of the file to remove.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns a negative error code value.</p> <p>For descriptions of all Install Aide API error codes, see "Install Aide Error Codes" on page 1004.</p>
Comments	PltRemoveInstallFile() removes the specified file from the user's handheld-install directory that is associated with files of the type to remove.
Compatibility	<p>Install Aide version: All. In version 4.0 if PltRemoveInstallFile() is unsuccessful, it returns a ERR_PALM_FILE_DELETE_FAILED error code. In earlier versions, this function did not return an error message if the specified file did not exist.</p> <p>Palm OS version: All.</p>

Install Aide API

PltRemoveInstallRegistry

PltRemoveInstallRegistry Function

Purpose	Removes the install conduit settings that are set by PltSetInstallRegistry() .
Declared In	<code>InstAide.h</code>
Prototype	<code>int PltRemoveInstallRegistry ()</code>
Parameters	None.
Returns	If successful, returns 0. If unsuccessful, returns a negative error code value. For descriptions of all Install Aide API error codes, see “ Install Aide Error Codes ” on page 1004.
Comments	This function does not affect third-party install conduits. It removes only those settings set by PltSetInstallRegistry() .
Compatibility	Install Aide version: 4.1 or later. Palm OS version: All.
See Also	PltSetInstallRegistry()

PltResetInstallMask Function

Purpose	Deselects the specified install conduit to run for the specified user during the next HotSync operation.
Declared In	<code>InstAide.h</code>
Prototype	<code>int PltResetInstallMask (TCHAR *pUser, DWORD dwMask)</code>
Parameters	<p>→ <i>pUser</i> The user name.</p> <p>→ <i>dwMask</i> A one-bit mask associated with the install conduit you do not want to run.</p>
Returns	If successful, returns 0. If unsuccessful, returns a negative error code value. For descriptions of all Install Aide API error codes, see “ Install Aide Error Codes ” on page 1004.
Comments	During a HotSync operation, HotSync Manager XORs the <i>dwMask</i> bit mask value you provide (corresponding to the install conduit you wish not to run for the specified user) with the bit mask values for all of the install conduits stored in the configuration entries. The resulting value is stored in the “Install” configuration entry for the specified user. The effect of this function is to clear the bit in the “Install” entry so that the corresponding install conduit does not run for the specified user during the next HotSync operation. If you do not know the <i>dwMask</i> value, use PltGetInstallConduitInfo() to get the <i>dwMask</i> value for each install conduit.
Compatibility	Install Aide version: All. Palm OS version: All.
See Also	PltIsInstallMaskSet() , PltGetInstallConduitInfo()

Install Aide API

PltSetInstallRegistry

PltSetInstallRegistry Function

Purpose	Restores install conduit information to default values.
Declared In	<code>InstAide.h</code>
Prototype	<code>int PltSetInstallRegistry ()</code>
Parameters	None.
Returns	If successful, returns 0. If unsuccessful, returns a negative error code value. For descriptions of all Install Aide API error codes, see “ Install Aide Error Codes ” on page 1004.
Comments	<code>PltSetInstallRegistry()</code> restores the default configuration entries for the install conduits that ships with HotSync Manager. This function is similar to <code>CmRestoreHotSyncSettings()</code> , except that it affects only the install conduits that ship with HotSync Manager. <code>PltRemoveInstallRegistry()</code> removes the settings that this function sets.
Compatibility	Install Aide version: All. Palm OS version: All.
See Also	<code>CmRestoreHotSyncSettings()</code> , <code>PltRemoveInstallRegistry()</code>

PltSetPath Function

Purpose	Sets the value of one of the Palm OS® Desktop path variables.
Declared In	InstAide.h
Prototype	<pre>int PltSetPath (unsigned short sPathType, TCHAR *pPathBuffer)</pre>
Parameters	<p>→ <i>sPathType</i> The path to set. Specify one of the defined paths described in “Path Definitions” on page 956.</p> <p>→ <i>pPathBuffer</i> The new path value to set in the configuration entries.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns a negative error code value.</p> <p>For descriptions of all Install Aide API error codes, see “Install Aide Error Codes” on page 1004.</p>
Comments	PltSetPath() sets the value of a path variable in the configuration entries.
Compatibility	Install Aide version: All. Palm OS version: All.
See Also	PltGetInstallFileFilter() , PltGetPath()

Install Aide API

Install Aide Error Codes

Install Aide Error Codes

[Table 15.1](#) lists the values of error codes that Install Aide API functions can return. Some function descriptions state which errors each function can return; some don't because they can return any of these error codes.

All of the named error codes below are defined as preprocessor constants, which are declared in the `InstAide.h` header file.

Table 15.1 **Install Aide API error codes**

Value	Error Code	Description
-1	—	A nonspecific error occurred.
0	—	The function was successful.
0xFFFFFD43	ERR_IA_INVALID_FILE_TYPE	The Install Aide does not recognize the file type.
0xFFFFFDD4	ERR_PALM_USER_ACCESS	The Install Aide cannot access the user data.
0xFFFFFDD5	ERR_PILOT_USER_ACCESS (Deprecated. Use ERR_PALM_USER_ACCESS instead.)	—
0xFFFFFDD5	ERR_PALM_BAD_FILENAME	The specified filename is invalid.
0xFFFFFDD6	ERR_PILOT_BAD_FILENAME (Deprecated. Use ERR_PALM_BAD_FILENAME instead.)	—
0xFFFFFDD6	ERR_PALM_NOT_FOUND	The specified user name could not be found in the database.
0xFFFFFDD7	ERR_PILOT_NOT_FOUND (Deprecated. Use ERR_PALM_NOT_FOUND instead.)	—
0xFFFFFDD7	ERR_PALM_NO_USERS	—

Table 15.1 Install Aide API error codes (*continued*)

Value	Error Code	Description
0xFFFFFDD8	ERR_PALM_NO_USER_FILE, ERR_PILOT_NO_USER_FILE (Deprecated. Use ERR_PALM_NO_USER_FILE instead.)	No users data file found.
0xFFFFFDD9	ERR_PALM_NO_DIRECTORY, ERR_PILOT_NO_DIRECTORY (Deprecated. Use ERR_PALM_NO_DIRECTORY instead.)	No directory was found for the specified user.
0xFFFFFDA	ERR_USER_MANAGER_BASE	No function returns this value. It is only the offset from which several error codes are based.
0xFFFFFD4	ERR_PALM_FILE_MOVE_FAILED	The Install Aide failed to move the specified install file because, for example, the file does not exist.
0xFFFFDF5	ERR_PALM_FILE_DELETE_FAILED	The Install Aide failed to remove the specified install file— for example, the file does not exist.
0xFFFFDF6	ERR_IA_INVALID_USER_ID	The user ID passed to <u>P1mGetUserNameFromID()</u> is invalid.
0xFFFFDF7	ERR_PALM_OTHER_USERSDAT_ACCESS_PROBLEM	A miscellaneous error occurred while trying to access the users data file.
0xFFFFDF8	ERR_PALM_SEMAPHORE_ACCESS	A sharing problem occurred with the users data file.

Install Aide API

Install Aide Error Codes

Table 15.1 Install Aide API error codes (*continued*)

Value	Error Code	Description
0xFFFFFD9	ERR_PALM_UNABLE_TO_CREATE_NEW_FILE	Creating a new users data file failed. The users data file did not exist; the Install Aide attempted, but failed to create one.
0xFFFFDFA	ERR_PALM_NO_CORE_PATH	A value for HotSync Manager's Core\Path configuration entry does not exist. The Install Aide cannot retrieve user information without it.
0xFFFFDFB	ERR_PILOT_INVALID_CREATORID	An invalid unique ID was passed to this function.
0xFFFFDFC	ERR_PILOT_INVALID_BUFFER	A buffer parameter is NULL.
0xFFFFDFD	ERR_PILOT_INVALID_FILE_TYPE	A file parameter is not a valid file type for use with the function.
0xFFFFDFE	ERR_PILOT_INVALID_INDEX	An index parameter exceeds the maximum value allowed for the index.
0xFFFFDFF	ERR_PILOT_INVALID_SOURCE_FILE	A file passed to the install function is not a valid file or was not found.

Table 15.1 Install Aide API error codes (*continued*)

Value	Error Code	Description
0xFFFFFE00	ERR_PILOT_FILE_ALREADY_EXISTS	An attempt was made to install a file that is already queued in the user's install directory on the desktop.
0xFFFFFE01	ERR_PILOT_INVALID_PATH	The path parameter passed to the PltSetPath() function is NULL.
0xFFFFFE02	ERR_PILOT_INVALID_REGISTRY	The requested path information is not stored in the configuration entries. Running the Palm OS® Desktop installation program often corrects this problem.
0xFFFFFE03	ERR_PILOT_INVALID_PATH_TYPE	The path type parameter value is not a valid path type value. For information about the path type constants, see " Path Definitions " on page 956.
0xFFFFFE04	ERR_PILOT_USERSDAT_ALREADY_EXISTS	No function returns this error code.
0xFFFFFE05	ERR_PILOT_INVALID_FILE_INDEX	The value of the file index parameter passed in is greater than the number of files.
0xFFFFFE06	ERR_PILOT_INVALID_FILENAME	The filename parameter value in a call to an install function is NULL.

Install Aide API

Install Aide Error Codes

Table 15.1 Install Aide API error codes (*continued*)

Value	Error Code	Description
0xFFFFFE07	ERR_IA_INVALID_USER, ERR_PILOT_INVALID_USER	The user name parameter value is either NULL or is not a valid user name.
0xFFFFFE08	ERR_PILOT_COPY_FAILED	The Install Aide could not copy the file.
0xFFFFFE09	ERR_PILOT_NO_USERSDAT_FILE	There is no users data file.
0xFFFFFE0A	ERR_PILOT_BUFSIZE_TOO_SMALL	The buffer supplied as a parameter to a function is either too small to contain the information that the function attempted to store or is invalid.
0xFFFFFE0B	ERR_PILOT_INVALID_USER_INDEX	The value of the index parameter is greater than the number of users.
0xFFFFFE0C	ERR_PILOT_NO_USERS	The users data file does not contain any user information.

User Data API

The User Data API accesses the [user data store](#) on the desktop computer. The user data store contains name, synchronization preferences, directory, and password information for all HotSync users created by the current Windows user.

The User Data functions are available in `UserData.dll` and declared in `UserData.h`.

The sections in this chapter are:

User Data API Versions	1009
User Data API Constants	1011
User Data API Functions	1015
User Data API Error Codes	1088

For more information, see “[Using the User Data API](#)” on page 145 in *C/C++ Sync Suite Companion*.

User Data API Versions

The User Data API continues to evolve with new functions and new versions of existing functions. Beginning with version 4.0, each version of the User Data API has a major version number and a minor version number. You can determine the version of the User Data API that you are using by calling the [UmGetLibVersion\(\)](#) function, available in User Data API versions 4.0 and later. Earlier versions do not have the `UmGetLibVersion()` function—which indicates that version 4.0 and later functionality is not present.

The User Data API maintains backward compatibility within a major version. The minor version number changes when new functions are added or bugs are fixed. This document includes version information for each function.

If your application depends on functions that are available only in certain versions of the User Data API, you need to determine the

User Data API

User Data API Versions

version of the User Data API with which you are dealing on a specific installation. To do so, call the [UmGetLibVersion\(\)](#) function, which returns both the major version number and minor version number of the User Data API on the desktop computer.

For example, if your application depends on a function that was added in version 4.0 of the User Data API, you need to call `UmGetLibVersion()` and then verify that the major number is 4 or greater and that the minor version number is 0 or greater.

For a summary of new features and other changes in the User Data API, see [Appendix A, “Revision History,”](#) on page 1105.

User Data API Constants

This section describes the following enumerated constants and groups of preprocessor constants that you use with the User Data API.

<u>UmUserSyncAction</u>	Defines a user's preferences for the type of synchronization operation to perform for a specified conduit.
<u>Palm OS Version Section and Key Names</u>	Define the section and key names in the user data store that specify where the handheld's Palm OS version number is stored.
<u>Version of the User Data API</u>	Defines the User Data API version that <u>UmGetLibVersion()</u> passes back.

UmUserSyncAction Enum

Purpose	Defines a user's preferences for the type of synchronization operation to perform for a specified conduit.
Declared In	<code>UserData.h</code>
Constants	<p><code>Synchronize = 0</code></p> <p>Perform a mirror-image synchronization between the desktop computer and the handheld.</p> <p><code>PCToHH</code></p> <p>Perform a restore from the desktop computer: overwrite the database on the handheld with the database on the desktop computer.</p> <p><code>HHToPC</code></p> <p>Perform a restore from the handheld: overwrite the desktop database with the database on the handheld.</p> <p><code>DoNothing</code></p> <p>Do not exchange data between the handheld and the desktop computer. HotSync Manager does, however, load the conduit, which can set flags, log messages, or do other housekeeping as necessary.</p> <p><code>Custom</code></p> <p>Perform any custom actions implemented in the conduit. HotSync Manager passes only this flag to the conduit, which must determine what action to take.</p>
Compatibility	User Data version: All. Palm OS version: All.
See Also	Um GetUserPermSyncPreferences() , Um GetUserTempSyncPreferences() , Um SetUserPermSyncPreferences() , Um SetUserTempSyncPreferences()

Palm OS Version Section and Key Names

Purpose	Define the section and key names in the user data store that specify where the handheld's Palm OS version number is stored.
Declared In	<code>UserData.h</code>
Constants	<pre>#define DEVICE_INFO_SECTION_NAME "Device Information" Specifies the section of the user data store that contains information about the user's handheld. #define DEVICE_ROM_VERSION "RomVersion" Specifies the key in the DEVICE_INFO_SECTION_NAME section whose value defines the version of Palm OS on the user's handheld.</pre>
Comments	HotSync Manager versions 6.0 and later retrieve the version of Palm OS from the user's handheld and store it in the user data store so that you can retrieve it outside of a HotSync operation. To retrieve this value, call UmGetInteger() and specify the above section and key names. Then use the SYNCROMVMAJOR() and SYNCROMVMINOR() macros to decode the version number.
Compatibility	User Data version: versions later than 4.2. Palm OS version: All.
See Also	UmGetString() , UmSetString()

User Data API

Version of the User Data API

Version of the User Data API

Purpose	Defines the User Data API version that UmGetLibVersion() passes back.
Declared In	<code>UserData.h</code>
Constants	<code>#define UM_LIB_VER_MAJOR 4</code> Indicates the major version number. <code>#define UM_LIB_VER_MINOR 2</code> Indicates the minor version number.
Comments	Version 4.1 or later of the User Data API defines these values, which UmGetLibVersion() passes back. In version 4.0, the first version in which UmGetLibVersion() is available, these constants were not defined, though UmGetLibVersion() returned 4 for the major version number and 0 for the minor version number.
Compatibility	User Data version: 4.1 or later. Palm OS version: All.
See Also	UmGetLibVersion()

User Data API Functions

This section describes the following functions, which applications and conduits can use to access the user data store.

<u>UmAddUser ()</u>	Adds a user to the user data store.
<u>UmClearInstallMask ()</u>	Deselects the specified install conduit to run for the specified user during the next HotSync operation.
<u>UmDeleteKey ()</u>	Deletes a key or an entire section from the specified user's area of the user data store.
<u>UmDeleteUser ()</u>	Deletes a user from the user data store.
<u>UmDeleteUserPermSyncPreferences ()</u>	Deletes the permanent synchronization preferences for <i>all</i> of the specified user's conduits.
<u>UmDeleteUserTempSyncPreferences ()</u>	Deletes the temporary synchronization preferences for <i>all</i> of the specified user's conduits.
<u>UmGetIDFromName ()</u>	Retrieves a user ID given the user's name.
<u>UmGetIDFromPath ()</u>	Retrieves a user ID given the user directory.
<u>UmGetInteger ()</u>	Retrieves an integer value from a key in the specified user's area of the user data store.
<u>UmGetLibVersion ()</u>	Retrieves the version of the User Data API.
<u>UmGetRootDirectory ()</u>	Retrieves the path of all user directories on the desktop computer (as stored in the Core\Path configuration entry).

User Data API

User Data API Functions

[UmGetString\(\)](#)

Retrieves a string value from a key in the specified user's area of the user data store.

[Um GetUserCount\(\)](#)

Returns the number of users in the user data store.

[Um GetUserDirectory\(\)](#)

Retrieves the user directory's name for the specified user ID.

[Um GetUserID\(\)](#)

Returns a user ID from the user data store by index.

[Um GetUserName\(\)](#)

Retrieves a user name in the user data store given a user ID.

[Um GetUserPassword\(\)](#)

Retrieves the encrypted user password for the specified user ID.

[Um GetUserPermSyncPreferences\(\)](#)

Retrieves a conduit's permanent synchronization preferences for the specified user ID.

[Um GetUserTempSyncPreferences\(\)](#)

Retrieves a conduit's temporary synchronization preferences for the specified user ID.

[Um IsInstallMaskSet\(\)](#)

Determines whether the specified install conduit is set to run for the specified user during the next HotSync operation.

[Um IsUserInstalled\(\)](#)

Determines whether the specified user is "installed."

[Um IsUserNameValid\(\)](#)

Determines whether the supplied string is a valid HotSync user name.

[Um IsUserProfile\(\)](#)

Determines whether an account is a user profile.

[Um RemoveUserTempSyncPreferences\(\)](#)

Removes the specified conduit's temporary synchronization preferences for the specified user ID.

<u>UmSetInstallMask()</u>	Selects the specified install conduit to run for the specified user during the next HotSync operation.
<u>UmSetInteger()</u>	Sets an integer value to a key in the specified user's area of the user data store.
<u>UmSetString()</u>	Sets a string value to a key in the specified user's area of the user data store.
<u>UmSetUserDirectory()</u>	Sets the directory name of the specified user ID.
<u>UmSetUserInstall()</u>	Sets or clears the "installed" flag of the specified user ID.
<u>UmSetUserName()</u>	Sets the user name of the specified user ID.
<u>UmSetUserPermSyncPreferences()</u>	Sets a conduit's permanent synchronization preferences for the specified user ID.
<u>UmSetUserTempSyncPreferences()</u>	Sets a conduit's temporary synchronization preferences for the specified user ID.
<u>UmSlotGetDisplayName()</u>	Retrieves the display name for the given slot on the specified user's handheld.
<u>UmSlotGetExpMgrVersion()</u>	Retrieves the saved version of the handheld's Expansion Manager for the specified user.
<u>UmSlotGetInfo()</u>	Retrieves the slot IDs for each of the expansion slots present on the specified user's handheld.
<u>UmSlotGetInstallDirectory()</u>	Retrieves the slot-install directory name (not the full path) for the specified user and handheld slot.

User Data API

User Data API Functions

[UmSlotGetMediaType \(\)](#)

Retrieves the media type of the given slot on the specified user's handheld.

[UmSlotGetSlotCount \(\)](#)

Retrieves the number of expansion slots on the handheld for the specified user.

UmAddUser Function

Purpose Adds a user to the user data store.

Declared In UserData.h

Prototype long UmAddUser (const char **pUser*,
 BOOL *bProfileUser*)

Parameters → *pUser*
A pointer to a character buffer that specifies the name of the user to add.

→ *bProfileUser*
If this is true, this method creates a new [user profile](#) named *pUser*. If false, it creates a regular user account.

Returns If successful, returns 0. If the user data store does not exist, this function creates it, adds the user, and returns 0 if successful.

If unsuccessful, returns one of the following error code values:

ERR_UM_BAD_FILENAME

ERR_UM_INVALID_USER_NAME

ERR_UM_NO_CORE_PATH

ERR_UM_NO_DIRECTORY

ERR_UM_NO_USER_FOUND

ERR_UM_NO_USERS

ERR_UM_NO_USERSDAT_FILE

ERR_UM_OTHER_USERSDAT_ACCESS_PROBLEM

ERR_UM_SAVE_ERR

ERR_UM_SEMAPHORE_ACCESS

ERR_UM_UNABLE_TO_CREATE_NEW_FILE

ERR_UM_USER_ACCESS

ERR_UM_USER_ALREADY_EXISTS

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

User Data API

UmAddUser

- Comments** Call [UmIsUserNameValid\(\)](#) to validate a user name before passing it into this function.
- Compatibility** User Data version: All.
Palm OS version: All.
- See Also** [UmIsUserNameValid\(\)](#), [UmIsUserProfile\(\)](#),
[UmDeleteUser\(\)](#)

UmClearInstallMask Function

Purpose Deselects the specified install conduit to run for the specified user during the next HotSync operation.

Declared In `UserData.h`

Prototype

```
long UmClearInstallMask (DWORD dwUserID,  
                         DWORD dwMask)
```

Parameters

→ *dwUserID*
The user ID, which specifies the user to reference in the user data store.

→ *dwMask*
A one-bit mask associated with the install conduit you do not want to run.

Returns If successful, returns 0.

If unsuccessful, returns one of the following error code values:

`ERR UM BAD FILENAME`
`ERR UM INVALID USER`
`ERR UM NO CORE PATH`
`ERR UM NO DIRECTORY`
`ERR UM NO USER FOUND`
`ERR UM NO USERS`
`ERR UM NO USERSDAT FILE`
`ERR UM OTHER USERSDAT ACCESS PROBLEM`
`ERR UM SEMAPHORE ACCESS`
`ERR UM UNABLE TO CREATE NEW FILE`
`ERR UM USER ACCESS`

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

User Data API

UmClearInstallMask

Comments	During a HotSync operation, HotSync Manager XORs the <i>dwMask</i> bit mask value you provide (corresponding to the install conduit you wish not to run for the specified user) with the bit mask values for all of the install conduits stored in the configuration entries. The resulting value is stored in the “Install” configuration entry for the specified user. The effect of this function is to clear the bit in the “Install” entry so that the corresponding install conduit does not run for the specified user during the next HotSync operation. If you do not know the <i>dwMask</i> value, use PltGetInstallConduitInfo() to get the <i>dwMask</i> value for each install conduit.
Compatibility	User Data version: All. Palm OS version: All.
See Also	UmGetUserID() , UmSetInstallMask() , UmIsInstallMaskSet()

UmDeleteKey Function

Purpose	Deletes a key or an entire section from the specified user's area of the user data store.
Declared In	<code>UserData.h</code>
Prototype	<pre>long UmDeleteKey (DWORD dwUserID, const TCHAR *pszSection, const TCHAR *pszKey)</pre>
Parameters	<p>→ <i>dwUserID</i> The user ID, which specifies the user to reference in the user data store.</p> <p>→ <i>pszSection</i> The section name in the specified user's area of the user data store.</p> <p>→ <i>pszKey</i> The key to delete. If this is NULL, then the entire section is deleted.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error code values. Specifically, the function returns</p> <p><code>ERR UM CANNOT WRITE TO STORE</code> if it cannot delete the key or section.</p> <p><code>ERR UM BAD FILENAME</code> <code>ERR UM BUFSIZE TOO SMALL</code> <code>ERR UM CANNOT WRITE TO STORE</code> <code>ERR UM INVALID POINTER</code> <code>ERR UM INVALID USER</code> <code>ERR UM NO CORE PATH</code> <code>ERR UM NO DIRECTORY</code> <code>ERR UM NO USER FOUND</code> <code>ERR UM NO USERS</code> <code>ERR UM NO USERSDAT FILE</code> <code>ERR UM OTHER USERSDAT ACCESS PROBLEM</code> <code>ERR UM SEMAPHORE ACCESS</code></p>

User Data API

UmDeleteKey

ERR_UM_UNABLE_TO_CREATE_NEW_FILE

ERR_UM_USER_ACCESS

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

Compatibility User Data version: All.
Palm OS version: All.

See Also [UmGetUserID\(\)](#)

UmDeleteUser Function

Purpose	Deletes a user from the user data store.
Declared In	<code>UserData.h</code>
Prototype	<code>long UmDeleteUser (DWORD dwUserID)</code>
Parameters	<p>$\rightarrow dwUserID$ The user ID, which specifies the user to reference in the user data store.</p>
Returns	<p>If successful, returns 0. If unsuccessful, returns one of the following error code values:</p> <p><code>ERR UM NO USERSDAT FILE</code> <code>ERR UM INVALID USER</code> <code>ERR UM NO USERS</code> <code>ERR UM SAVE ERR</code> <code>ERR UM OTHER USERSDAT ACCESS PROBLEM</code> <code>ERR UM SEMAPHORE ACCESS</code> <code>ERR UM UNABLE TO CREATE NEW FILE</code> <code>ERR UM NO CORE PATH</code> <code>ERR UM INVALID POINTER</code> <code>ERR UM USER ALREADY EXISTS</code> <code>ERR UM USER DIR ALREADY IN USE</code> <code>ERR UM UNABLE TO CREATE NEW FILE</code></p> <p>For descriptions of all User Data API error codes, see “User Data API Error Codes” on page 1088.</p>
Compatibility	User Data version: All. Palm OS version: All.
See Also	UmGetUserID() , UmAddUser()

User Data API

UmDeleteUserPermSyncPreferences

UmDeleteUserPermSyncPreferences Function

Purpose	Deletes the permanent synchronization preferences for <i>all</i> of the specified user's conduits.
Declared In	UserData.h
Prototype	long UmDeleteUserPermSyncPreferences (DWORD dwUserID, DWORD dwCreatorID)
Parameters	<p>→ <i>dwUserID</i> The user ID, which specifies the user to reference in the user data store.</p> <p>→ <i>dwCreatorID</i> This value is ignored.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>ERR UM BAD FILENAME ERR UM INVALID USER ERR UM NO CORE PATH ERR UM NO DIRECTORY ERR UM NO USER FOUND ERR UM NO USERS ERR UM NO USERSDAT FILE ERR UM OTHER USERSDAT ACCESS PROBLEM ERR UM SAVE ERR ERR UM SEMAPHORE ACCESS ERR UM UNABLE TO CREATE NEW FILE ERR UM USER ACCESS</p> <p>For descriptions of all User Data API error codes, see “User Data API Error Codes” on page 1088.</p>

Comments	UmDeleteUserPermSyncPreferences () clears the permanent synchronization preference for <i>all</i> conduits. The result is the same as if the user has never clicked HotSync Manager's Custom > Change option and altered any permanent synchronization preferences.
Compatibility	User Data version: All. Palm OS version: All.
See Also	UmGetUserID() , UmSetUserPermSyncPreferences() , Um GetUserPermSyncPreferences()

User Data API

UmDeleteUserTempSyncPreferences

UmDeleteUserTempSyncPreferences Function

Purpose	Deletes the temporary synchronization preferences for <i>all</i> of the specified user's conduits.
Declared In	UserData.h
Prototype	<pre>long UmDeleteUserTempSyncPreferences (DWORD dwUserID, DWORD dwCreatorID)</pre>
Parameters	<p>→ <i>dwUserID</i> The user ID, which specifies the user to reference in the user data store.</p> <p>→ <i>dwCreatorID</i> This value is ignored.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>ERR UM BAD FILENAME ERR UM INVALID USER ERR UM NO CORE PATH ERR UM NO DIRECTORY ERR UM NO USER FOUND ERR UM NO USERS ERR UM NO USERSDAT FILE ERR UM OTHER USERSDAT ACCESS PROBLEM ERR UM SAVE ERR ERR UM SEMAPHORE ACCESS ERR UM UNABLE TO CREATE NEW FILE ERR UM USER ACCESS</p> <p>For descriptions of all User Data API error codes, see “User Data API Error Codes” on page 1088.</p>

Comments

`UmDeleteUserTempSyncPreferences()` clears the temporary synchronization preference for *all* conduits so that the actions set in the conduits' permanent synchronization preferences will be taken during the next HotSync operation. The result is the same as if the user has never clicked HotSync Manager's **Custom > Change** option and altered any temporary synchronization preferences.

NOTE: This function clears *all* conduits' temporary synchronization preferences. Contrast it with [UmRemoveUserTempSyncPreferences\(\)](#), which clears the temporary preferences for only *one* conduit.

Compatibility

User Data version: All.

Palm OS version: All.

See Also

[UmGetUserID\(\)](#), [UmSetUserTempSyncPreferences\(\)](#),
[Um GetUserTempSyncPreferences\(\)](#),
[UmRemoveUserTempSyncPreferences\(\)](#)

User Data API

UmGetIDFromName

UmGetIDFromName Function

Purpose	Retrieves a user ID given the user's name.
Declared In	UserData.h
Prototype	long UmGetIDFromName (const TCHAR * <i>pszName</i> , DWORD * <i>pdwUserID</i>)
Parameters	 → <i>pszName</i> A pointer to a character buffer that contains the user's name. ← <i>pdwUserID</i> A buffer to receive the user ID.
Returns	If successful, returns 0. If unsuccessful, returns one of the following error code values: ERR_UM_BAD_FILENAME ERR_UM_BUFSIZE_TOO_SMALL ERR_UM_INVALID_POINTER ERR_UM_NO_CORE_PATH ERR_UM_NO_DIRECTORY ERR_UM_NO_USER_FOUND ERR_UM_NO_USERS ERR_UM_NO_USERSDAT_FILE ERR_UM_OTHER_USERSDAT_ACCESS_PROBLEM ERR_UM_SEMAPHORE_ACCESS ERR_UM_UNABLE_TO_CREATE_NEW_FILE ERR_UM_USER_ACCESS For descriptions of all User Data API error codes, see " User Data API Error Codes " on page 1088.

Comments	Note that it is possible for the user data store to contain the same name more than once. Because the user ID is the only value that the User Data API ensures is unique, each instance of the same name has a different user ID. Therefore, in that case, you must perform additional checking to determine whether the user name is unique.
Compatibility	User Data version: All. Palm OS version: All.
See Also	UmGetUserID()

UmGetIDFromPath Function

Purpose	Retrieves a user ID given the user directory.
Declared In	<code>UserData.h</code>
Prototype	<code>long UmGetIDFromPath (const TCHAR *pszPath, DWORD *pdwUserID)</code>
Parameters	<p>→ <i>pszPath</i> A pointer to a character buffer that contains the path to the user directory.</p> <p>← <i>pdwUserID</i> A buffer to receive the user ID.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p><code>ERR UM BAD FILENAME</code> <code>ERR UM BUFSIZE TOO SMALL</code> <code>ERR UM INVALID POINTER</code> <code>ERR UM INVALID USER</code> <code>ERR UM INVALID USER DIR</code> <code>ERR UM NO CORE PATH</code> <code>ERR UM NO DIRECTORY</code> <code>ERR UM NO USER FOUND</code> <code>ERR UM NO USERS</code> <code>ERR UM NO USERSDAT FILE</code> <code>ERR UM OTHER USERSDAT ACCESS PROBLEM</code> <code>ERR UM SEMAPHORE ACCESS</code> <code>ERR UM UNABLE TO CREATE NEW FILE</code> <code>ERR UM USER ACCESS</code></p> <p>For descriptions of all User Data API error codes, see “User Data API Error Codes” on page 1088.</p>

Compatibility User Data version: All.
Palm OS version: All.

See Also [UmGetUserID\(\)](#)

User Data API

UmGetInteger

UmGetInteger Function

Purpose	Retrieves an integer value from a key in the specified user's area of the user data store.
Declared In	UserData.h
Prototype	<pre>long UmGetInteger (DWORD dwUserID, const TCHAR *pszSection, const TCHAR *pszKey, long *pBuf, long lDefault)</pre>
Parameters	<p>→ <i>dwUserID</i> The user ID, which specifies the user to reference in the user data store.</p> <p>→ <i>pszSection</i> The section name in the specified user's area of the user data store.</p> <p>→ <i>pszKey</i> The key of the integer to retrieve.</p> <p>← <i>pBuf</i> The buffer to receive the integer.</p> <p>→ <i>lDefault</i> The default integer to return if no integer can be retrieved for the specified key.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error code values and sets <i>pBuf</i> to the value specified by <i>lDefault</i> . ERR_UM_BAD_FILENAME ERR_UM_BUFSIZE_TOO_SMALL ERR_UM_INVALID_POINTER ERR_UM_INVALID_USER ERR_UM_NO_CORE_PATH ERR_UM_NO_DIRECTORY ERR_UM_NO_USER_FOUND ERR_UM_NO_USERS ERR_UM_NO_USERSDAT_FILE

ERR_UM_OTHER_USERSDAT_ACCESS_PROBLEM

ERR_UM_SEMAPHORE_ACCESS

ERR_UM_UNABLE_TO_CREATE_NEW_FILE

ERR_UM_USER_ACCESS

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

Comments

Use this function to read the value of any integer key in the user data store provided the name of the section and key. You can use this function to read keys that you create. However, if a User Data API function exists for reading a value, use the specific function rather than this general-purpose function.

HotSync Manager versions 6.0 and later retrieve the version of Palm OS from the user’s handheld and store it in the user data store.

Because the User Data API provides no function specifically for reading the Palm OS version number, to do so you must use this function and pass in the section and key names described in “[Palm OS Version Section and Key Names](#)” on page 1013.

Compatibility

User Data version: All.

Palm OS version: All.

See Also

[UmGetUserID\(\)](#), [UmSetInteger\(\)](#)

User Data API

UmGetLibVersion

UmGetLibVersion Function

Purpose	Retrieves the version of the User Data API.
Declared In	UserData.h
Prototype	<pre>long UmGetLibVersion (DWORD *pdwMajor, DWORD *pdwMinor)</pre>
Parameters	<p>$\leftarrow pdwMajor$ The major version number, which is defined by UM_LIB_VER_MAJOR.</p> <p>$\leftarrow pdwMinor$ The minor version number, which is defined by UM_LIB_VER_MINOR.</p>
Returns	Always returns 0.
Comments	Call UmGetLibVersion() to determine the version of the User Data API that you are using before calling any of its functions. For more information, see “ User Data API Versions ” on page 1009. For details about what changes between versions, see Appendix A, “Revision History,” on page 1105. The version number constants that this function passes back are defined in “ Version of the User Data API ” on page 1014.
Compatibility	User Data version: 4.0 or later. Palm OS version: All.

UmGetRootDirectory Function

Purpose Retrieves the path of all user directories on the desktop computer (as stored in the [Core\Path](#) configuration entry).

Declared In UserData.h

Prototype

```
long UmGetRootDirectory (TCHAR *pRootDirBuffer,  
                         short *psRootDirBufSize)
```

Parameters

← *pRootDirBuffer*
A pointer to a character buffer in which to pass back the path.

← *psRootDirBufSize*
A pointer to a short in which to pass back the size of the path.

Returns If successful, returns a value >0 that is the length of the root path. The same length value is also stored in *psRootDirBuffer*.

If no root path is found, returns 0.

If unsuccessful, returns one of the following error codes. If the buffer is not large enough to contain the root path, this function sets *psRootDirBufSize* to the required size and returns the error `ERR_UM_BUFSIZE_TOO_SMALL`.

`ERR_UM_BUFSIZE_TOO_SMALL`

`ERR_UM_INVALID_POINTER`

`ERR_UM_NO_CORE_PATH`

`ERR_UM_NO_USERS`

`ERR_UM_NO_USERSDAT_FILE`

`ERR_UM_OTHER_USERSDAT_ACCESS_PROBLEM`

`ERR_UM_SEMAPHORE_ACCESS`

`ERR_UM_UNABLE_TO_CREATE_NEW_FILE`

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

User Data API

UmGetRootDirectory

Comments	UmGetRootDirectory() retrieves the value stored in the Core\Path configuration entry. Because HotSync Manager versions 6.0 and later are aware of multiple Windows users, each Windows user must have a separate Core\Path value. To meet Windows standards for the placement of user data, a typical value is C:\Documents and Settings\<WinUsername>\My Documents\Palm OS Desktop. If you need the full path of a HotSync user's directory, then call Um GetUserDirectory() and append that path to the Core\Path value.
Compatibility	User Data version: All. Palm OS version: All.
See Also	Um GetUserDirectory() , Um SetUserDirectory()

UmGetString Function

Purpose Retrieves a string value from a key in the specified user's area of the user data store.

Declared In `UserData.h`

Prototype

```
long UmGetString (DWORD dwUserID,
                  const TCHAR *pszSection, const TCHAR *pszKey,
                  TCHAR *pBuf, long *lSize,
                  const TCHAR *pszDefault)
```

Parameters

→ `dwUserID`
The user ID, which specifies the user to reference in the user data store.

→ `pszSection`
The section name in the specified user's area of the user data store.

→ `pszKey`
The key of the string to retrieve.

← `pBuf`
The buffer to receive the string.

↔ `lSize`
A pointer to a long value that specifies the size, in bytes, of the buffer referenced by `pBuf`. Upon return, this value is the actual size of `pBuf`. If this function fails because the buffer was too small, the value of `lSize` upon return is the required size of the buffer to hold the string.

→ `pszDefault`
The default string to return if no string can be retrieved for the specified key.

Returns If successful, returns 0.

If unsuccessful, returns one of the following error codes. If the buffer is not large enough to contain the string, this function sets `lSize` to the required size and returns the error code `ERR UM_BUFSIZE_TOO_SMALL`. If the string could not be retrieved, `pszDefault` is returned in `pBuf`.

`ERR UM_BAD_FILENAME`

`ERR UM_BUFSIZE_TOO_SMALL`

User Data API

UmGetString

```
ERR_UM_INVALID_POINTER  
ERR_UM_INVALID_USER  
ERR_UM_NO_CORE_PATH  
ERR_UM_NO_DIRECTORY  
ERR_UM_NO_USER_FOUND  
ERR_UM_NO_USERS  
ERR_UM_NO_USERSDAT_FILE  
ERR_UM_OTHER_USERSDAT_ACCESS_PROBLEM  
ERR_UM_SEMAPHORE_ACCESS  
ERR_UM_STRING_TOO_BIG  
ERR_UM_UNABLE_TO_CREATE_NEW_FILE  
ERR_UM_USER_ACCESS
```

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

Comments	Use this function to read the value of any string key in the user data store provided the name of the section and key. You can use this function to read keys that you create. However, if a User Data API function exists for reading a value, use the specific function rather than this general-purpose function.
Compatibility	User Data version: All. Palm OS version: All.
See Also	UmGetUserID() , UmSetString()

UmGetUserCount Function

Purpose	Returns the number of users in the user data store.
Declared In	<code>UserData.h</code>
Prototype	<code>short UmGetUserCount (void)</code>
Parameters	None.
Returns	If successful, returns a value ≥ 0 that is the number of users in the user data store. If unsuccessful, returns a value < 0 that is one of the following error code values: <code>ERR UM NO CORE PATH</code> <code>ERR UM NO USERS</code> <code>ERR UM NO USERSDAT FILE</code> <code>ERR UM OTHER USERSDAT ACCESS PROBLEM</code> <code>ERR UM SEMAPHORE ACCESS</code> <code>ERR UM UNABLE TO CREATE NEW FILE</code>
	For descriptions of all User Data API error codes, see “ User Data API Error Codes ” on page 1088.
Compatibility	User Data version: All. Palm OS version: All.

User Data API

UmGetUserDirectory

UmGetUserDirectory Function

Purpose	Retrieves the user directory's name for the specified user ID.
Declared In	UserData.h
Prototype	<pre>long UmGetUserDirectory (DWORD dwUserID, TCHAR *pUserDirBuffer, short *psUserDirBufSize)</pre>
Parameters	<p>→ <i>dwUserID</i> The user ID, which specifies the user to reference in the user data store.</p> <p>← <i>pUserDirBuffer</i> A pointer to a TCHAR buffer. Upon return, this buffer contains the directory name of the specified user.</p> <p>↔ <i>psUserDirBufSize</i> A pointer to a short value that specifies the size, in TCHARs, of the buffer referenced by <i>pUserDirBuffer</i>. Upon return, this value is the actual size of the <i>pUserDirBuffer</i>. If this function fails because the buffer was too small, the value of <i>psUserDirBufSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns a value >=0 that is the length of the user directory name also stored in <i>pUserDirBuffer</i>.</p> <p>If unsuccessful, returns one of the following error codes. If the buffer is not large enough to contain the directory name, this function sets <i>psUserDirBufSize</i> to the required size and returns the error <code>ERR_UM_BUFSIZE_TOO_SMALL</code>.</p> <p>ERR_UM_BAD_FILENAME ERR_UM_BUFSIZE_TOO_SMALL ERR_UM_INVALID_POINTER ERR_UM_INVALID_USER ERR_UM_NO_CORE_PATH ERR_UM_NO_DIRECTORY ERR_UM_NO_USER_FOUND ERR_UM_NO_USERS ERR_UM_NO_USERSDAT_FILE</p>

ERR_UM_OTHER_USERSDAT_ACCESS_PROBLEM

ERR_UM_SEMAPHORE_ACCESS

ERR_UM_UNABLE_TO_CREATE_NEW_FILE

ERR_UM_USER_ACCESS

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

Comments To get a complete path, concatenate the result of [UmGetRootDirectory\(\)](#) and [Um GetUserDirectory\(\)](#)—for example, `root_directory = "C:\Documents and Settings\<WinUsername>\My Documents\Palm OS Desktop"` and `user_directory = "NUser"`.

Compatibility User Data version: All.
Palm OS version: All.

See Also [UmGetUserID\(\)](#), [UmSetUserDirectory\(\)](#)

User Data API

UmGetUserID

UmGetUserID Function

Purpose Returns a user ID from the user data store by index.

Declared In UserData.h

Prototype long UmGetUserID (short *sIndex*, DWORD **pdwUserID*)

Parameters

→ *sIndex*
A zero-based index that specifies a user in the user data store.
← *pdwUserID*
A pointer to a DWORD to receive a user ID.

Returns If successful, returns 0.

If unsuccessful, returns one of the following error code values:

ERR_UM_INVALID_POINTER
ERR_UM_INVALID_USER_INDEX
ERR_UM_NO_CORE_PATH
ERR_UM_NO_USERS
ERR_UM_NO_USERSDAT_FILE
ERR_UM_OTHER_USERSDAT_ACCESS_PROBLEM
ERR_UM_SEMAPHORE_ACCESS
ERR_UM_UNABLE_TO_CREATE_NEW_FILE

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

Compatibility User Data version: All. Versions earlier than 4.0 returned the user ID if successful, returned 0 if no user ID was found, and a negative value on error. Because the user ID value could be negative before it is cast, a successful result could also return a negative value. Version 4.0 of the User Data API fixes this problem, as shown in “Result” above.

Palm OS version: All.

See Also [UmGetIDFromPath\(\)](#), [UmGetIDFromName\(\)](#)

UmGetUserName Function

Purpose	Retrieves a user name in the user data store given a user ID.
Declared In	<code>UserData.h</code>
Prototype	<pre>long UmGetUserName (DWORD dwUserID, char *pUserNameBuffer, short *psUserNameBufSize)</pre>
Parameters	<p>→ <i>dwUserID</i> The user ID, which specifies the user to reference in the user data store.</p> <p>← <i>pUserNameBuffer</i> A pointer to a character buffer. Upon return, this buffer contains the name of the specified user.</p> <p>↔ <i>psUserNameBufSize</i> A pointer to a short value that specifies the size, in bytes, of the buffer referenced by <i>pUserNameBuffer</i>. Upon return, this value is the actual size of the <i>pUserNameBuffer</i>. If this function fails because the buffer was too small, the value of <i>psUserNameBufSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns a value >0 that is the length of the user name. The same length value is also stored in <i>pUserNameBuffer</i>.</p> <p>If unsuccessful, returns one of the following error code values. If the buffer is not large enough to contain the user name, this function sets <i>psUserNameBufSize</i> to the required size and returns the error <code>ERR_UM_BUFSIZE_TOO_SMALL</code>.</p> <p>ERR_UM_BAD_FILENAME ERR_UM_BUFSIZE_TOO_SMALL ERR_UM_INVALID_POINTER ERR_UM_NO_CORE_PATH ERR_UM_NO_DIRECTORY ERR_UM_NO_USER_FOUND ERR_UM_NO_USERS ERR_UM_NO_USERSDAT_FILE ERR_UM_OTHER_USERSDAT_ACCESS_PROBLEM</p>

User Data API

UmGetUserName

ERR_UM_SEMAPHORE_ACCESS
ERR_UM_UNABLE_TO_CREATE_NEW_FILE
ERR_UM_USER_ACCESS

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

Compatibility User Data version: All.
Palm OS version: All.

See Also [UmGetUserID\(\)](#), [UmSetUserName\(\)](#)

UmGetUserPassword Function

Purpose	Retrieves the encrypted user password for the specified user ID.
Declared In	<code>UserData.h</code>
Prototype	<pre>long UmGetUserPassword (DWORD dwUserID, TCHAR *pUserPasswordBuffer, short *psUserPasswordBufSize)</pre>
Parameters	<p>→ <i>dwUserID</i> The user ID, which specifies the user to reference in the user data store.</p> <p>← <i>pUserPasswordBuffer</i> A pointer to a character buffer. Upon return, this buffer contains the encrypted password of the specified user.</p> <p>↔ <i>psUserPasswordBufSize</i> A pointer to a short value that specifies the size, in TCHARS, of the buffer referenced by <i>pUserPasswordBuffer</i>. Upon return, this value is the actual size of the <i>pUserPasswordBuffer</i>. If this function fails because the buffer was too small, the value of <i>psUserPasswordBufSize</i> upon return is the required size of the buffer.</p>
Returns	<p>If successful, returns a value ≥ 0 that is the length of the password. The same length value is also stored in <i>pUserPasswordBuffer</i>.</p> <p>If unsuccessful, returns a value < 0 that is one of the following nonzero error code values. If the buffer is not large enough to contain the user name, this function sets <i>psUserPasswordBufSize</i> to the required size and returns the error <code>ERR_UM_BUFSIZE_TOO_SMALL</code>.</p> <p>ERR_UM_BAD_FILENAME ERR_UM_BUFSIZE_TOO_SMALL ERR_UM_INVALID_POINTER ERR_UM_INVALID_USER ERR_UM_NO_CORE_PATH ERR_UM_NO_DIRECTORY ERR_UM_NO_USER_FOUND</p>

User Data API

UmGetUserPassword

ERR_UM_NO_USERS
ERR_UM_NO_USERSDAT_FILE
ERR_UM_OTHER_USERSDAT_ACCESS_PROBLEM
ERR_UM_SEMAPHORE_ACCESS
ERR_UM_UNABLE_TO_CREATE_NEW_FILE
ERR_UM_USER_ACCESS

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

Compatibility	User Data version: All. See “ User Data API, Version 4.1 Changes ” on page 1118 for more information. Palm OS version: All.
See Also	UmGetUserID()

Um GetUserPermSyncPreferences Function

Purpose	Retrieves a conduit's permanent synchronization preferences for the specified user ID.
Declared In	<code>UserData.h</code>
Prototype	<pre>long Um GetUserPermSyncPreferences (DWORD dwUserID, DWORD dwCreatorID, long *pUsaAction)</pre>
Parameters	<p>→ <i>dwUserID</i> The user ID, which specifies the user to reference in the user data store.</p> <p>→ <i>dwCreatorID</i> The creator ID of the conduit to access.</p> <p>← <i>pUsaAction</i> A pointer to a <code>long</code> to receive the synchronization preferences defined by UmUserSyncAction.</p>
Returns	<p>If successful, returns one of the UmUserSyncAction that is also passed back in <i>pUsaAction</i>.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>-1 A nonspecific error occurred or no permanent preference has been set for the specified creator ID.</p> <p>ERR_UM_BAD_FILENAME ERR_UM_INVALID_USER ERR_UM_NO_CORE_PATH ERR_UM_NO_DIRECTORY ERR_UM_NO_USER_FOUND ERR_UM_NO_USERS ERR_UM_NO_USERSDAT_FILE ERR_UM_OTHER_USERSDAT_ACCESS_PROBLEM ERR_UM_SAVE_ERR ERR_UM_SEMAPHORE_ACCESS ERR_UM_UNABLE_TO_CREATE_NEW_FILE ERR_UM_USER_ACCESS</p>

User Data API

Um GetUserPermSyncPreferences

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

Compatibility User Data version: All.
Palm OS version: All.

See Also [UmGetUserID\(\)](#), [UmSetUserPermSyncPreferences\(\)](#),
[UmDeleteUserPermSyncPreferences\(\)](#)

Um GetUserTempSyncPreferences Function

Purpose Retrieves a conduit's temporary synchronization preferences for the specified user ID.

Declared In UserData.h

Prototype

```
long Um GetUserTempSyncPreferences (DWORD dwUserID,  
                                     DWORD dwCreatorID, long *pUsaAction)
```

Parameters

→ *dwUserID*
The user ID, which specifies the user to reference in the user data store.

→ *dwCreatorID*
The creator ID of the conduit to access.

← *pUsaAction*
A pointer to a long to receive the synchronization preferences defined by [UmUserSyncAction](#).

Returns If successful, returns one of the [UmUserSyncAction](#) that is also passed back in *pUsaAction*.

If unsuccessful, returns one of the following error code values:

-1

A nonspecific error occurred or no temporary preference has been set for the specified creator ID.

ERR_UM_BAD_FILENAME

ERR_UM_INVALID_USER

ERR_UM_NO_CORE_PATH

ERR_UM_NO_DIRECTORY

ERR_UM_NO_USER_FOUND

ERR_UM_NO_USERS

ERR_UM_NO_USERSDAT_FILE

ERR_UM_OTHER_USERSDAT_ACCESS_PROBLEM

ERR_UM_SAVE_ERR

ERR_UM_SEMAPHORE_ACCESS

ERR_UM_UNABLE_TO_CREATE_NEW_FILE

ERR_UM_USER_ACCESS

User Data API

UmGetUserTempSyncPreferences

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

Compatibility User Data version: All.
Palm OS version: All.

See Also [UmGetUserID\(\)](#), [UmSetUserTempSyncPreferences\(\)](#),
[UmRemoveUserTempSyncPreferences\(\)](#),
[UmDeleteUserTempSyncPreferences\(\)](#)

UmIsInstallMaskSet Function

Purpose Determines whether the specified install conduit is set to run for the specified user during the next HotSync operation.

Declared In UserData.h

Prototype

```
long UmIsInstallMaskSet (DWORD dwUserID,  
                         DWORD dwMask)
```

Parameters

→ *dwUserID*

The user ID, which specifies the user to reference in the user data store.

→ *dwMask*

A one-bit mask associated with the install conduit that you want to know the run status of.

Returns

If the specified install conduit is set to run, returns `true`.

If the specified install conduit is set *not* to run, returns `false`.

If unsuccessful, returns one of the following nonzero error code values:

`ERR UM BAD FILENAME`

`ERR UM INVALID USER`

`ERR UM NO CORE PATH`

`ERR UM NO DIRECTORY`

`ERR UM NO USER FOUND`

`ERR UM NO USERS`

`ERR UM NO USERSDAT FILE`

`ERR UM OTHER USERSDAT ACCESS PROBLEM`

`ERR UM SEMAPHORE ACCESS`

`ERR UM UNABLE TO CREATE NEW FILE`

`ERR UM USER ACCESS`

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

User Data API

UmIsInstallMaskSet

Comments	During a HotSync operation, HotSync Manager ANDs the <i>dwMask</i> bit mask value you provide (corresponding to the install conduit you wish to check the run status of) with the "Install" configuration entry for the specified user. This function returns true if the bit mask value you specified is set; otherwise it returns false. If you do not know the <i>dwMask</i> value, use PltGetInstallConduitInfo() to get the <i>dwMask</i> value for each install conduit.
Compatibility	User Data version: All. Palm OS version: All.
See Also	UmGetUserID() , UmSetInstallMask() , UmClearInstallMask()

UmIsUserInstalled Function

Purpose	Determines whether the specified user is “installed.”
Declared In	<code>UserData.h</code>
Prototype	<code>long UmIsUserInstalled (DWORD dwUserID)</code>
Parameters	<p>→ <i>dwUserID</i> The user ID, which specifies the user to reference in the user data store.</p>
Returns	<p>If the user is an installed user, returns true. If the user is not an installed user, returns false. If unsuccessful, returns one of the following error code values:</p> <p>ERR UM BAD FILENAME ERR UM INVALID USER ERR UM NO CORE PATH ERR UM NO DIRECTORY ERR UM NO USER FOUND ERR UM NO USERS ERR UM NO USERSDAT FILE ERR UM OTHER USERSDAT ACCESS PROBLEM ERR UM SEMAPHORE ACCESS ERR UM UNABLE TO CREATE NEW FILE ERR UM USER ACCESS</p> <p>For descriptions of all User Data API error codes, see “User Data API Error Codes” on page 1088.</p>

User Data API

UmIsUserInstalled

Comments	“Installed” users are those who have completed at least one HotSync operation so that their user ID is on both the desktop computer and a handheld. Users created on the desktop, but who have never completed a HotSync operation, are not “installed” users. They become “installed” users if they synchronize a handheld with the same user ID as is on the desktop.
Compatibility	User Data version: All. Palm OS version: All.
See Also	UmGetUserID() , UmSetUserInstall()

UmIsUserNameValid Function

Purpose	Determines whether the supplied string is a valid HotSync user name.
Declared In	<code>UserData.h</code>
Prototype	<code>bool UmIsUserNameValid (const TCHAR *pUserName)</code>
Parameters	<code>→ pUserName</code> A pointer to a character buffer that contains the user name to validate.
Returns	If this user name is valid, returns <code>true</code> . If this user name is invalid, returns <code>false</code> .
Comments	This function validates the following characteristics of the supplied user name: <ul style="list-style-type: none">Contains at least one but no more than 20 valid characters (a user name is invalid if <code>pUserName</code> is <code>NULL</code> or points to an empty string buffer).Contains none of these invalid characters: <code>~!@#\$%^&*()`+=[]{}:"';,<>?/ \ A user name may contain a period (.), hyphen (-), or underscore (_) character.</code>Contains at least one valid character that is not a space character.Does not begin with a multi-byte space character. A user name may begin with a single-byte space character; however, this may not be allowed in future versions of this function. Use this function to validate a user name before passing it to <u>UmAddUser()</u> to create a new user. Validating the name ensures that, when the user name appears in the HotSync log, the name contains no special HTML characters that would cause the log to be formatted incorrectly.
Compatibility	User Data version: versions later than 4.2. Palm OS version: All.
See Also	<u>UmAddUser()</u>

User Data API

UmIsUserProfile

UmIsUserProfile Function

Purpose	Determines whether an account is a user profile.
Declared In	UserData.h
Prototype	long UmIsUserProfile (DWORD dwUserID)
Parameters	$\rightarrow dwUserID$ The user ID, which specifies the user to reference in the user data store.
Returns	If this account is a user profile, returns true. If this account is a regular user account, returns false. If unsuccessful, returns one of the following error code values: ERR_UM_BAD_FILENAME ERR_UM_INVALID_USER ERR_UM_NO_CORE_PATH ERR_UM_NO_DIRECTORY ERR_UM_NO_USER_FOUND ERR_UM_NO_USERS ERR_UM_NO_USERSDAT_FILE ERR_UM_OTHER_USERSDAT_ACCESS_PROBLEM ERR_UM_SEMAPHORE_ACCESS ERR_UM_UNABLE_TO_CREATE_NEW_FILE ERR_UM_USER_ACCESS For descriptions of all User Data API error codes, see “ User Data API Error Codes ” on page 1088.
Comments	For more information on user profiles , see the glossary in the <i>Introduction to Conduit Development</i> .
Compatibility	User Data version: All. Palm OS version: All.
See Also	UmGetUserID() , UmIsUserProfile()

UmRemoveUserTempSyncPreferences Function

Purpose	Removes the specified conduit's temporary synchronization preferences for the specified user ID.
Declared In	<code>UserData.h</code>
Prototype	<code>long UmRemoveUserTempSyncPreferences (DWORD dwUserID, DWORD dwCreatorID)</code>
Parameters	<p>→ <code>dwUserID</code> The user ID, which specifies the user to reference in the user data store.</p> <p>→ <code>dwCreatorID</code> The creator ID of the conduit to access.</p>
Returns	If successful, returns 0. If unsuccessful, returns one of the following error code values: ERR_UM_BAD_FILENAME ERR_UM_INVALID_USER ERR_UM_NO_CORE_PATH ERR_UM_NO_DIRECTORY ERR_UM_NO_USER_FOUND ERR_UM_NO_USERS ERR_UM_NO_USERSDAT_FILE ERR_UM_OTHER_USERSDAT_ACCESS_PROBLEM ERR_UM_SAVE_ERR ERR_UM_SEMAPHORE_ACCESS ERR_UM_UNABLE_TO_CREATE_NEW_FILE ERR_UM_USER_ACCESS
	For descriptions of all User Data API error codes, see “ User Data API Error Codes ” on page 1088.

User Data API

UmRemoveUserTempSyncPreferences

Comments UmRemoveUserTempSyncPreferences () clears the temporary synchronization preferences for the specified conduit so that the action set in the permanent synchronization preferences will be taken during the next HotSync operation. The result is the same as if the user had never clicked HotSync Manager's **Custom > Change** option and altered the conduit's temporary synchronization preferences.

NOTE: This function clears only *one* conduit's temporary synchronization preferences. Contrast it with [UmDeleteUserTempSyncPreferences\(\)](#), which clears the temporary preferences for *all* the user's conduits.

Compatibility User Data version: All.
Palm OS version: All.

See Also [UmGetUserID\(\)](#), [UmSetUserTempSyncPreferences\(\)](#),
[UmGetUserTempSyncPreferences\(\)](#),
[UmDeleteUserTempSyncPreferences\(\)](#)

UmSetInstallMask Function

Purpose Selects the specified install conduit to run for the specified user during the next HotSync operation.

Declared In UserData.h

Prototype

```
long UmSetInstallMask (DWORD dwUserID,  
                      DWORD dwMask)
```

Parameters

→ *dwUserID*
The user ID, which specifies the user to reference in the user data store.

→ *dwMask*
A one-bit mask associated with the install conduit you want to run.

Returns If successful, returns 0.

If unsuccessful, returns one of the following error code values:

ERR UM BAD FILENAME
ERR UM INVALID USER
ERR UM NO CORE PATH
ERR UM NO DIRECTORY
ERR UM NO USER FOUND
ERR UM NO USERS
ERR UM NO USERSDAT FILE
ERR UM OTHER USERSDAT ACCESS PROBLEM
ERR UM SEMAPHORE ACCESS
ERR UM UNABLE TO CREATE NEW FILE
ERR UM USER ACCESS

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

User Data API

UmSetInstallMask

Comments	During a HotSync operation, HotSync Manager ORs the <i>dwMask</i> bit mask value you provide (corresponding to the install conduit you wish to run) with the "Install" configuration entry for the specified user. The resulting value is stored in the "Install" configuration entry for the specified user. The effect of this function is to set the bit in the "Install" entry so that the corresponding install conduit runs for the specified user during the next HotSync operation, without changing the state of any other bits in the "Install" entry. If you do not know the <i>dwMask</i> value, use PltGetInstallConduitInfo() to get the <i>dwMask</i> value for each install conduit.
Compatibility	User Data version: All. Palm OS version: All.
See Also	UmGetUserID() , UmIsInstallMaskSet() , UmClearInstallMask()

UmSetInteger Function

Purpose Sets an integer value to a key in the specified user's area of the user data store.

Declared In UserData.h

Prototype

```
long UmSetInteger (DWORD dwUserID,
                   const TCHAR *pszSection, const TCHAR *pszKey,
                   long lValue)
```

Parameters

→ *dwUserID*
The user ID, which specifies the user to reference in the user data store.

→ *pszSection*
The section name in the specified user's area of the user data store.

→ *pszKey*
The key of the integer to set.

→ *lValue*
The integer to write to the key in the specified user's area of the user data store.

Returns If successful, returns 0.

If unsuccessful, returns one of the following error code values:

ERR UM BAD FILENAME

ERR UM BUFSIZE TOO SMALL

ERR UM CANNOT WRITE TO STORE

ERR UM INVALID POINTER

ERR UM INVALID USER

ERR UM NO CORE PATH

ERR UM NO DIRECTORY

ERR UM NO USER FOUND

ERR UM NO USERS

ERR UM NO USERSDAT FILE

ERR UM OTHER USERSDAT ACCESS PROBLEM

ERR UM SEMAPHORE ACCESS

User Data API

UmSetInteger

ERR UM UNABLE TO CREATE NEW FILE

ERR UM USER ACCESS

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

Comments Use this function to set the value of any integer key in the user data store provided the name of the section and key. You can use this function to create or set your own keys. However, if a User Data API function exists for setting a value, use the specific function rather than this general-purpose function.

Compatibility User Data version: All.
Palm OS version: All.

See Also [UmGetUserID\(\)](#), [UmGetInteger\(\)](#)

UmSetString Function

Purpose Sets a string value to a key in the specified user's area of the user data store.

Declared In UserData.h

Prototype

```
long UmSetString (DWORD dwUserID,
                  const TCHAR *pszSection, const TCHAR *pszKey,
                  const TCHAR *pszValue)
```

Parameters

→ *dwUserID*
The user ID, which specifies the user to reference in the user data store.

→ *pszSection*
The section name in the specified user's area of the user data store.

→ *pszKey*
The key of the string to set.

→ *pszValue*
The string to write to the key in the specified user's area of the user data store.

Returns If successful, returns 0.

If unsuccessful, returns one of the following error code values:

ERR UM BAD FILENAME

ERR UM BUFSIZE TOO SMALL

ERR UM CANNOT WRITE TO STORE

ERR UM INVALID POINTER

ERR UM INVALID USER

ERR UM NO CORE PATH

ERR UM NO DIRECTORY

ERR UM NO USER FOUND

ERR UM NO USERS

ERR UM NO USERSDAT FILE

ERR UM OTHER USERSDAT ACCESS PROBLEM

ERR UM SEMAPHORE ACCESS

User Data API

UmSetString

ERR_UM_UNABLE_TO_CREATE_NEW_FILE

ERR_UM_USER_ACCESS

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

Comments Use this function to set the value of any string key in the user data store provided the name of the section and key. You can use this function to create or set your own keys. However, if a User Data API function exists for setting a value, use the specific function rather than this general-purpose function.

Compatibility User Data version: All.
Palm OS version: All.

See Also [UmGetUserID\(\)](#), [UmGetString\(\)](#)

UmSetUserDirectory Function

Purpose	Sets the directory name of the specified user ID.
Declared In	<code>UserData.h</code>
Prototype	<code>long UmSetUserDirectory (DWORD dwUserID, const TCHAR *pUserDir)</code>
Parameters	<p>→ <i>dwUserID</i> The user ID, which specifies the user to reference in the user data store.</p> <p>→ <i>pUserDir</i> A character buffer that contains the user directory name to set.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>ERR UM BAD FILENAME ERR UM INVALID USER ERR UM INVALID USER DIR ERR UM NO CORE PATH ERR UM NO DIRECTORY ERR UM NO USER FOUND ERR UM NO USERS ERR UM NO USERSDAT FILE ERR UM OTHER USERSDAT ACCESS PROBLEM ERR UM SAVE ERR ERR UM SEMAPHORE ACCESS ERR UM UNABLE TO CREATE NEW FILE ERR UM USER ACCESS ERR UM USER DIR ALREADY IN USE</p> <p>For descriptions of all User Data API error codes, see “User Data API Error Codes” on page 1088.</p>

User Data API

UmSetUserDirectory

Compatibility User Data version: All.
Palm OS version: All.

See Also [UmGetUserID\(\)](#), [UmGetUserDirectory\(\)](#)

UmSetUserInstall Function

Purpose	Sets or clears the “installed” flag of the specified user ID.
Declared In	<code>UserData.h</code>
Prototype	<code>long UmSetUserInstall (DWORD dwUserID, BOOL bUserInstall)</code>
Parameters	<p>→ <i>dwUserID</i> The user ID, which specifies the user to reference in the user data store.</p> <p>→ <i>bUserInstall</i> A Boolean value. If this is <code>true</code>, the user is set as an “installed” user; if this is <code>false</code>, the user is not set as an “installed” user.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p><code>ERR UM BAD FILENAME</code> <code>ERR UM INVALID USER</code> <code>ERR UM NO CORE PATH</code> <code>ERR UM NO DIRECTORY</code> <code>ERR UM NO USER FOUND</code> <code>ERR UM NO USERS</code> <code>ERR UM NO USERSDAT FILE</code> <code>ERR UM OTHER USERSDAT ACCESS PROBLEM</code> <code>ERR UM SAVE ERR</code> <code>ERR UM SEMAPHORE ACCESS</code> <code>ERR UM UNABLE TO CREATE NEW FILE</code> <code>ERR UM USER ACCESS</code></p>
	<p>For descriptions of all User Data API error codes, see “User Data API Error Codes” on page 1088.</p>

User Data API

UmSetUserInstall

Comments	“Installed” users are those who have completed at least one HotSync operation so that their user ID is on both the desktop computer and a handheld. Users created on the desktop, but who have never completed a HotSync operation, are not “installed” users. They become “installed” users if they synchronize a handheld with the same user ID as is on the desktop.
Compatibility	User Data version: All. Palm OS version: All.
See Also	UmGetUserID() , UmIsUserInstalled()

UmSetUserName Function

Purpose	Sets the user name of the specified user ID.
Declared In	<code>UserData.h</code>
Prototype	<code>long UmSetUserName (DWORD dwUserID, const char *pUserName)</code>
Parameters	<p>→ <i>dwUserID</i> The user ID, which specifies the user to reference in the user data store.</p> <p>→ <i>*pUserName</i> A character buffer that contains the user name to set.</p>
Returns	<p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>ERR UM BAD FILENAME ERR UM INVALID USER ERR UM NO CORE PATH ERR UM NO DIRECTORY ERR UM NO USER FOUND ERR UM NO USERS ERR UM NO USERSDAT FILE ERR UM OTHER USERSDAT ACCESS PROBLEM ERR UM SAVE ERR ERR UM SEMAPHORE ACCESS ERR UM UNABLE TO CREATE NEW FILE ERR UM UNABLE TO CREATE NEW FILE ERR UM USER ACCESS ERR UM USER ALREADY EXISTS</p> <p>For descriptions of all User Data API error codes, see “User Data API Error Codes” on page 1088.</p>

User Data API

UmSetUserName

Compatibility User Data version: All.
Palm OS version: All.

See Also [UmGetUserID\(\)](#), [UmGetUserName\(\)](#)

UmSetUserPermSyncPreferences Function

Purpose Sets a conduit's permanent synchronization preferences for the specified user ID.

Declared In `UserData.h`

Prototype `long UmSetUserPermSyncPreferences (DWORD dwUserID,
DWORD dwCreatorID, long usaAction)`

Parameters

- `dwUserID`
The user ID, which specifies the user to reference in the user data store.
- `dwCreatorID`
The creator ID of the conduit to access.
- `usaAction`
The synchronization action to perform. Specify one of the [UmUserSyncAction](#) enum values.

Returns If successful, returns `true`.

If the preferences could not be set, returns `false`.

If unsuccessful, returns one of the following error code values:

- `ERR UM BAD FILENAME`
- `ERR UM INVALID USER`
- `ERR UM NO CORE PATH`
- `ERR UM NO DIRECTORY`
- `ERR UM NO USER FOUND`
- `ERR UM NO USERS`
- `ERR UM NO USERSDAT FILE`
- `ERR UM OTHER USERSDAT ACCESS PROBLEM`
- `ERR UM SAVE ERR`
- `ERR UM SEMAPHORE ACCESS`
- `ERR UM UNABLE TO CREATE NEW FILE`
- `ERR UM USER ACCESS`

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

User Data API

UmSetUserPermSyncPreferences

Compatibility User Data version: All.
Palm OS version: All.

See Also [UmGetUserID\(\)](#), [UmGetUserPermSyncPreferences\(\)](#),
[UmDeleteUserPermSyncPreferences\(\)](#)

UmSetUserTempSyncPreferences Function

Purpose Sets a conduit's temporary synchronization preferences for the specified user ID.

Declared In `UserData.h`

Prototype `long UmSetUserTempSyncPreferences (DWORD dwUserID,
DWORD dwCreatorID, long usaAction)`

Parameters

- `dwUserID`
The user ID, which specifies the user to reference in the user data store.
- `dwCreatorID`
The creator ID of the conduit to access.
- `usaAction`
The synchronization action to perform. Specify one of the [UmUserSyncAction](#) enum values.

Returns If successful, returns `true`.

If the preferences could not be set, returns `false`.

If unsuccessful, returns one of the following error code values:

- `ERR UM BAD FILENAME`
- `ERR UM INVALID USER`
- `ERR UM NO CORE PATH`
- `ERR UM NO DIRECTORY`
- `ERR UM NO USER FOUND`
- `ERR UM NO USERS`
- `ERR UM NO USERSDAT FILE`
- `ERR UM OTHER USERSDAT ACCESS PROBLEM`
- `ERR UM SAVE ERR`
- `ERR UM SEMAPHORE ACCESS`
- `ERR UM UNABLE TO CREATE NEW FILE`
- `ERR UM USER ACCESS`

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

User Data API

UmSetUserTempSyncPreferences

Compatibility User Data version: All.
Palm OS version: All.

See Also [UmGetUserID\(\)](#), [UmGetUserTempSyncPreferences\(\)](#),
[UmRemoveUserTempSyncPreferences\(\)](#),
[UmDeleteUserTempSyncPreferences\(\)](#)

UmSlotGetDisplayName Function

Purpose Retrieves the display name for the given slot on the specified user's handheld.

Declared In UserData.h

Prototype

```
long UmSlotGetDisplayName (DWORD dwUserId,  
                           DWORD dwSlotId, TCHAR *pszSlotDisplayName,  
                           long *plSize)
```

Parameters

→ *dwUserId*
The ID of the user.

→ *dwSlotId*
The ID of the slot for which to get the name. To get slot IDs, use the User Data API's [UmSlotGetInfo\(\)](#) function.

← *pszSlotDisplayName*
Pointer to a character buffer that holds the display name of the specified slot. The calling function must allocate this buffer first.

↔ *plSize*
Pointer to a long that indicates the size of the *pszSlotDisplayName* buffer. Upon entry, it points to the size allocated for the buffer. Upon exit, if the size passed in was too small, it points to the number of bytes you must allocate for the buffer.

Returns If successful, returns 0.

If unsuccessful, returns one of the following error code values:

ERR_UM_INVALID_POINTER

ERR_UM_BUFSIZE_TOO_SMALL

ERR_UM_DEV_CFG_DATA_NOT_AVAILABLE

If the *pszSlotDisplayName* is too small to hold the display name, this function returns ERR_UM_BUFSIZE_TOO_SMALL and passes back the required size in *plSize*.

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

User Data API

UmSlotGetDisplayName

- Comments** Use the display name, not the slot ID, to identify the slot for the user's benefit.
HotSync Manager assigns names to slots based on their media type at the beginning of each HotSync operation and saves it for the corresponding user in the user information store on the desktop. This function simply passes back the saved information. Therefore it may not be accurate for the next HotSync operation, because the user may have changed or updated the handheld.
You must call [UmSlotGetExpMgrVersion\(\)](#) to check for the presence of the Expansion Manager (and its version) on the handheld before calling this or any of the other slot-related User Data function.
-
- Example**
- ```
TCHAR szDisplayName [256];
long lSize = sizeof (szDisplayName);
long retval = UmSlotGetDisplayName (dwUserId, dwSlotId,
 szDisplayName, &lSize);
```
- 
- Compatibility** User Data version: 4.0 or later.  
Palm OS version: 4.0 or later.
- See Also** [UmSlotGetInfo\(\)](#)

## UmSlotGetExpMgrVersion Function

**Purpose** Retrieves the saved version of the handheld's Expansion Manager for the specified user.

**Declared In** UserData.h

**Prototype**

```
long UmSlotGetExpMgrVersion (DWORD dwUserId,
 DWORD *pdwExpMgrVersion)
```

**Parameters**

→ *dwUserId*  
The ID of the user.

← *pdwExpMgrVersion*  
A pointer to a DWORD to hold the version of the Expansion Manager on the handheld. If the Expansion Manager is not present, this value is 0—in which case, do not call other slot-related functions.

**Returns** If successful, returns 0.

If unsuccessful, returns one of the following error code values:

ERR UM INVALID POINTER

ERR UM DEV CFG DATA NOT AVAILABLE

For descriptions of all User Data API error codes, see “[User Data API Error Codes](#)” on page 1088.

**Comments** HotSync Manager retrieves this information from the handheld at the beginning of each HotSync operation and saves it for the corresponding user in the user information store on the desktop. This function simply returns the saved value. Therefore this value may not be accurate for the next HotSync operation, because the user may have changed or updated the handheld.

You must call `UmSlotGetExpMgrVersion()` to check for the presence of the Expansion Manager (and its version) on the handheld before calling any of the other slot-related User Data functions.

---

**Example**

```
DWORD dwExpMgrVersion;
long retval = UmSlotGetExpMgrVersion (dwUserId,
 &dwExpMgrVersion);
```

---

**Compatibility** User Data version: 4.0 or later.  
Palm OS version: 4.0 or later.

## User Data API

### UmSlotGetInfo

---

## UmSlotGetInfo Function

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Retrieves the slot IDs for each of the expansion slots present on the specified user's handheld.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Declared In</b> | UserData.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Prototype</b>   | <pre>long UmSlotGetInfo (DWORD dwUserId,<br/>                    DWORD *pdwSlotIdList, WORD *pwNumEntries)</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Parameters</b>  | <p>→ <i>dwUserId</i><br/>The ID of the user.</p> <p>← <i>pdwSlotIdList</i><br/>A pointer to a DWORD array to hold the ID for each slot.</p> <p>↔ <i>pwNumEntries</i><br/>Pointer to a WORD that indicates the number of entries in the <i>pdwSlotIdList</i> array. Upon entry, it points to the number of entries allocated for the <i>pdwSlotIdList</i> array. The caller should first call <a href="#">UmSlotGetSlotCount()</a> to get the slot count and allocate that many entries. Upon exit, it points to the number of slots for which IDs were successfully retrieved—which might not be the same as the actual number of slots present.</p> |
| <b>Returns</b>     | <p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p>ERR_UM_INVALID_POINTER<br/>ERR_UM_DEV_CFG_DATA_NOT_AVAILABLE<br/>ERR_UM_BUFSIZE_TOO_SMALL</p> <p>If the <i>pdwSlotIdList</i> array is too small, this function returns ERR_UM_BUFSIZE_TOO_SMALL and sets <i>pwNumEntries</i> to the required size.</p> <p>For descriptions of all User Data API error codes, see “<a href="#">User Data API Error Codes</a>” on page 1088.</p>                                                                                                                                                           |

**Comments** The Expansion Manager on the handheld identifies slots by slot reference numbers. These slot reference numbers may change depending on the order in which slot drivers are loaded by the Expansion Manager. Moreover, slot reference numbers are available only to conduits during a HotSync operation. Therefore the User Data API uses slot IDs to identify slots instead.

HotSync Manager assigns slot IDs to slots on the handheld at the beginning of each HotSync operation and saves them for the corresponding user in the user information store on the desktop. This function simply returns the saved information. Therefore it may not be accurate for the next HotSync operation because the user may have changed or updated the handheld.

You must call [UmSlotGetExpMgrVersion\(\)](#) to check for the presence of the Expansion Manager (and its version) on the handheld before calling this or any of the other slot-related User Data function.

---

**Example**

```
WORD wNumSlots;
DWORD *pdwSlotIdList;
long retval = UmSlotGetSlotCount (dwUserId, &wNumSlots);
if (retval == ERROR_SUCCESS)
{
 pdwSlotIdList = new DWORD [wNumSlots];
 if (pdwSlotIdList)
 {
 UmSlotGetSlotInfo (dwUserId, pdwSlotIdList,
 wNumSlots);
 }
}
```

---

**Compatibility**

User Data version: 4.0 or later.

Palm OS version: 4.0 or later.

**See Also**

[UmSlotGetSlotCount\(\)](#), [UmSlotGetExpMgrVersion\(\)](#)

## User Data API

### *UmSlotGetInstallDirectory*

---

## UmSlotGetInstallDirectory Function

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Retrieves the slot-install directory name (not the full path) for the specified user and handheld slot.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Declared In</b> | <code>UserData.h</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Prototype</b>   | <pre>long UmSlotGetInstallDirectory (DWORD dwUserId,<br/>                                DWORD dwSlotId, TCHAR *pszSlotInstallDir,<br/>                                long *plSize)</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Parameters</b>  | <p>→ <i>dwUserId</i><br/>The ID of the user.</p> <p>→ <i>dwSlotId</i><br/>The ID of the slot for which to get the slot-install directory. To get slot IDs, use the User Data API's <a href="#">UmSlotGetInfo()</a> function.</p> <p>← <i>pszSlotInstallDir</i><br/>Pointer to a character buffer that holds the name of the slot-install directory of the specified slot. The calling function must allocate this buffer first.</p> <p>↔ <i>plSize</i><br/>Pointer to a long that indicates the size of the <i>pszSlotInstallDir</i> buffer. Upon entry, it points to the size allocated for the buffer. Upon exit, if the size passed in was too small, it points to the number of bytes you must allocate for the buffer.</p> |
| <b>Returns</b>     | <p>If successful, returns 0.</p> <p>If unsuccessful, returns one of the following error code values:</p> <p><code>ERR_UM_INVALID_POINTER</code><br/><code>ERR_UM_BUFSIZE_TOO_SMALL</code><br/><code>ERR_UM_DEV_CFG_DATA_NOT_AVAILABLE</code></p> <p>If the <i>pszSlotInstallDir</i> is too small to hold the display name, this function returns <code>ERR_UM_BUFSIZE_TOO_SMALL</code> and passes back the required size in <i>plSize</i>.</p> <p>For descriptions of all User Data API error codes, see “<a href="#">User Data API Error Codes</a>” on page 1088.</p>                                                                                                                                                          |

**Comments** The slot-install directory is the location on the desktop where the Install Aide places files queued to be installed on the corresponding slot on the handheld for the specified user.

HotSync Manager saves this information at the beginning of each HotSync operation and puts it for the corresponding user in the user information store on the desktop. This function simply returns the saved information. Therefore it may not be accurate for the next HotSync operation because the user may have changed or updated the handheld. If files are queued to be installed to a slot and the slot information changes during the next HotSync operation, an install conduit may ignore those files but does not remove the slot-install directory itself.

You must call [UmSlotGetExpMgrVersion\(\)](#) to check for the presence of the Expansion Manager (and its version) on the handheld before calling this or any of the other slot-related User Data function.

---

**Example**

```
TCHAR szInstallDir [256];
long lSize = sizeof (szInstallDir);
long retval = UmSlotGetInstallDirectory (dwUserId, dwSlotId,
 szInstallDir, &lSize);
```

---

**Compatibility** User Data version: 4.0 or later.  
Palm OS version: 4.0 or later.

**See Also** [UmSlotGetInfo\(\)](#), [UmSlotGetExpMgrVersion\(\)](#), “[Install Directory Terminology](#)” on page 131 in the *C/C++ Sync Suite Companion*

## User Data API

### *UmSlotGetMediaType*

---

## UmSlotGetMediaType Function

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Retrieves the media type of the given slot on the specified user's handheld.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Declared In</b> | <code>UserData.h</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Prototype</b>   | <code>long UmSlotGetMediaType (DWORD dwUserId,<br/>                        DWORD dwSlotId,  DWORD *pdwSlotMediaType)</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Parameters</b>  | <p>→ <i>dwUserId</i><br/>The ID of the user.</p> <p>→ <i>dwSlotId</i><br/>The ID of the slot for which to get the media type. To get slot IDs, use the User Data API's <a href="#">UmSlotGetInfo()</a> function.</p> <p>← <i>pdwSlotMediaType</i><br/>Pointer to a DWORD that holds the media type of the specified slot. Slot media types are defined in “<a href="#">Media Type Constants</a>” on page 552.</p>                                                                                                                                                                                        |
| <b>Returns</b>     | If successful, returns 0.<br><br>If unsuccessful, returns one of the following error code values:<br><br><code>ERR UM INVALID POINTER</code><br><code>ERR UM DEV CFG DATA NOT AVAILABLE</code><br><br>For descriptions of all User Data API error codes, see “ <a href="#">User Data API Error Codes</a> ” on page 1088.                                                                                                                                                                                                                                                                                 |
| <b>Comments</b>    | HotSync Manager retrieves this information from the handheld at the beginning of each HotSync operation and saves it for the corresponding user in the user information store on the desktop. This function simply returns the saved information. Therefore it may not be accurate for the next HotSync operation because the user may have changed or updated the handheld.<br><br>You must call <a href="#">UmSlotGetExpMgrVersion()</a> to check for the presence of the Expansion Manager (and its version) on the handheld before calling this or any of the other slot-related User Data function. |
| <b>Example</b>     | <pre>DWORD dwSlotMediaType;<br/>long retval = <b>UmSlotGetMediaType</b> (dwUserId, dwSlotId,<br/>                               &amp;dwSlotMediaType);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                             |

**Compatibility** User Data version: 4.0 or later.  
Palm OS version: 4.0 or later.

**See Also** [UmSlotGetInfo\(\)](#), [UmSlotGetExpMgrVersion\(\)](#)

## User Data API

### *UmSlotGetSlotCount*

---

## UmSlotGetSlotCount Function

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Retrieves the number of expansion slots on the handheld for the specified user.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Declared In</b> | UserData.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Prototype</b>   | long UmSlotGetSlotCount (DWORD <i>dwUserId</i> ,<br>WORD * <i>pwNumSlots</i> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Parameters</b>  | <p>→ <i>dwUserId</i><br/>The ID of the user.</p> <p>← <i>pwNumSlots</i><br/>A pointer to a DWORD to hold the number of slots on the handheld. If the handheld has no expansion slots, the value it points to is 0.</p>                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Returns</b>     | If successful, returns 0.<br><br>If unsuccessful, returns one of the following error code values:<br><br>ERR_UM_INVALID_POINTER<br><br>ERR_UM_DEV_CFG_DATA_NOT_AVAILABLE<br><br>If the handheld has no slots, this function returns<br>ERR_UM_DEV_CFG_DATA_NOT_AVAILABLE and passes back<br>* <i>pwNumSlots</i> with a value of 0.<br><br>For descriptions of all User Data API error codes, see “ <a href="#">User Data API Error Codes</a> ” on page 1088.                                                                                                                                             |
| <b>Comments</b>    | HotSync Manager retrieves this information from the handheld at the beginning of each HotSync operation and saves it for the corresponding user in the user data store on the desktop. This function simply passes back the saved value. Therefore this value may not be accurate for the next HotSync operation, because the user may have changed or updated the handheld.<br><br>You must call <a href="#">UmSlotGetExpMgrVersion()</a> to check for the presence of the Expansion Manager (and its version) on the handheld before calling this or any of the other slot-related User Data function. |
| <b>Example</b>     | <pre>WORD wNumSlots; long retval = <b>UmSlotGetSlotCount</b> (dwUserId, &amp;wNumSlots);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

**Compatibility** User Data version: 4.0 or later.  
Palm OS version: 4.0 or later.

**See Also** [UmSlotGetExpMgrVersion\(\)](#)

## User Data API

### User Data API Error Codes

---

## User Data API Error Codes

[Table 16.1](#) lists the values of error codes that User Data API functions can return. The description of each function states which errors each function can return.

All of the named error codes below are defined as preprocessor constants, which are declared in the `UserData.h` header file.

**Table 16.1 User Data API error codes**

| Value      | Error Code                        | Description                                                                                                                                           |
|------------|-----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0xFFFFFFFF | —                                 | A nonspecific error occurred.                                                                                                                         |
| 0xFFFFFDDE | ERR_UM_NO_DEFAULT_USER            | No default HotSync user is set or an invalid user ID is set. This error code is reserved for future use.                                              |
| 0xFFFFDDEE | ERR_UM_MULTIPLE_USERS_EXISTS      | There is more than one user in the user data store. This error code is reserved for future use.                                                       |
| 0xFFFFDDEF | ERR_UM_DEV_CFG_DATA_NOT_AVAILABLE | The requested information about the handheld is not available—for example, no expansion slot information was previously saved for the specified user. |
| 0xFFFFDF0  | ERR_UM_FUNCTION_NOT_SUPPORTED     | This function is no longer supported.                                                                                                                 |
| 0xFFFFDF1  | ERR_UM_STRING_TOO_BIG             | The string to retrieve is too large to store.                                                                                                         |

**Table 16.1 User Data API error codes (*continued*)**

| <b>Value</b>          | <b>Error Code</b>                    | <b>Description</b>                                                                                       |
|-----------------------|--------------------------------------|----------------------------------------------------------------------------------------------------------|
| 0xFFFFFD <sup>2</sup> | ERR_UM_SYNC_PATH_TOO_BIG             | No function returns this error code.                                                                     |
| 0xFFFFFD <sup>3</sup> | ERR_UM_CANNOT_WRITE_TO_STORE         | The attempt to write to the specified user's area of the user data store failed.                         |
| 0xFFFFFD <sup>4</sup> | ERR_UM_USER_DIR_ALREADY_IN_USE       | The specified user directory name is already in use by another user.                                     |
| 0xFFFFFD <sup>5</sup> | ERR_UM_USER_ALREADY_EXISTS           | The specified user already exists.                                                                       |
| 0xFFFFFD <sup>6</sup> | ERR_UM_INVALID_POINTER               | The specified pointer is invalid.                                                                        |
| 0xFFFFFD <sup>7</sup> | ERR_UM_NO_CORE_PATH                  | A value for the Core\Path configuration entry does not exist. See <a href="#">UmGetRootDirectory()</a> . |
| 0xFFFFFD <sup>8</sup> | ERR_UM_UNABLE_TO_CREATE_NEW_FILE     | Creating a new file failed.                                                                              |
| 0xFFFFFD <sup>9</sup> | ERR_UM_SEMAPHORE_ACCESS              | The specified user index is invalid.                                                                     |
| 0xFFFFFD <sup>A</sup> | ERR_UM_OTHER_USERSDAT_ACCESS_PROBLEM | An unknown error occurred while trying to access the user data store.                                    |
| 0xFFFFFD <sup>B</sup> | ERR_UM_BASE                          | No function returns this error code.                                                                     |

## User Data API

### User Data API Error Codes

---

**Table 16.1 User Data API error codes (*continued*)**

| <b>Value</b> | <b>Error Code</b>        | <b>Description</b>                                                       |
|--------------|--------------------------|--------------------------------------------------------------------------|
| 0xFFFFFDFFC  | ERR_UM_SAVE_ERR          | Saving changes was not successfully completed.                           |
| 0xFFFFFDFFD  | ERR_UM_USER_ACCESS       | The User Data function cannot access the user data.                      |
| 0xFFFFFDFFE  | ERR_UM_BAD_FILENAME      | The file passed to the User Data function is not a valid file type.      |
| 0xFFFFFDFF   | ERR_UM_NOT_FOUND         | The user name parameter is not already contained in the user data store. |
| 0xFFFFFE00   | ERR_UM_NO_USER_FOUND     | The user does not exist in the user data store.                          |
| 0xFFFFFE01   | ERR_UM_NO_DIRECTORY      | The user directory does not exist.                                       |
| 0xFFFFFE02   | ERR_UM_INVALID_USER_DIR  | The specified user directory is invalid.                                 |
| 0xFFFFFE03   | ERR_UM_INVALID_USER_NAME | The specified user name is invalid.                                      |
| 0xFFFFFE04   | ERR_UM_INVALID_BUFFER    | No function returns this error code.                                     |
| 0xFFFFFE05   | ERR_UM_INVALID_INDEX     | No function returns this error code.                                     |
| 0xFFFFFE06   | ERR_UM_INVALID_REGISTRY  | A User Data function tried to write an invalid configuration entry.      |

**Table 16.1 User Data API error codes (*continued*)**

| <b>Value</b> | <b>Error Code</b>              | <b>Description</b>                                                                                                                                              |
|--------------|--------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0xFFFFFE07   | ERR_UM_INVALID_USER            | The supplied user ID is NULL or is not for an available user.                                                                                                   |
| 0xFFFFFE08   | ERR_UM_BUFSIZE_TOO_SMALL       | The supplied buffer is too small to contain the information requested.                                                                                          |
| 0xFFFFFE09   | ERR_UM_INVALID_USER_INDEX      | The user index that was passed in is out of range. This value should be greater than or equal to zero and less than the number of users in the user data store. |
| 0xFFFFFE0A   | ERR_UM_USERSDAT_ALREADY_EXISTS | No function returns this error code.                                                                                                                            |
| 0xFFFFFE0B   | ERR_UM_NO_USERS                | The user data store exists, but contains no information.                                                                                                        |
| 0xFFFFFE0C   | ERR_UM_NO_USERSDAT_FILE        | No user data store was found. Most of the functions fail in this case. Call <a href="#">UmAddUser()</a> to create the store.                                    |

## User Data API

### *User Data API Error Codes*

---

# Password Library

The Password Library enables external applications to verify whether a user-supplied, nonencrypted string matches the stored, encrypted password.

This function is provided as a library function in PWD5.lib and is declared in password.h.

Password Library Functions . . . . . 1093

# Password Library Functions

This section describes the one function in the Password Library, [PwdVerify\(\)](#).

## Password Library

### PwdVerify

---

## PwdVerify Function

**Purpose** Verifies that a user-supplied, unencrypted string matches the stored, encrypted password.

**Declared In** password.h

**Prototype** BOOL PwdVerify (char \**sEncryptedPwd*,  
const char \**sPassword*)

**Parameters**

- *sEncryptedPwd*  
A pointer to a null-terminated character buffer containing the encrypted password to verify against.
- *sPassword*  
A pointer to null-terminated character buffer containing the nonencrypted password to verify.

**Returns** If the *sPassword* value (after PwdVerify() encrypts it) matches the *sEncryptedPwd* value, returns true.

If the values do not match, returns false.

---

**NOTE:** PwdVerify() is case-sensitive. It detects a match only if the strings match, including case.

---

**Comments** The user can set a password on the device to control access to private records and to unlock the device. During a HotSync® session, HotSync Manager compares the encrypted password on the device with the encrypted password stored in the users data file. If they do not match (or if the users data file contains no password for the user), HotSync Manager updates the file with the password on the device. On the desktop, an application can use the same password to control access to the user's data.

For example, if an application prompts the user to enter a password in a dialog box, it must call [Um GetUser Password\(\)](#) to retrieve the user's encrypted password from the users data file. The application then calls PwdVerify() to verify that the user supplied the correct password. PwdVerify() encrypts the user-supplied, unencrypted string (*sPassword*) and compares it to the stored, encrypted password (*sEncryptedPwd*) retrieved by [Um GetUser Password\(\)](#). If PwdVerify() returns true, the user supplied the correct password.

**Compatibility**

**NOTE:** Palm OS Cobalt does not allow the user's password on the device to be transferred to the desktop as described above. Instead such users can set an independent password using the Palm OS® Desktop software, which you can verify against using `PwdVerify()`.

---

In Palm OS 4.0 and later, passwords on the device are case-sensitive. Therefore `PwdVerify()` returns `true` only if the user-supplied string is identical, including case, to the stored password.

However, in Palm OS versions earlier than 4.0, the Security application on the device converts all uppercase characters in a password to lowercase characters when the user sets a password. Therefore for the desktop's `PwdVerify()` function to detect a match with the converted password, the caller must ensure that the user-supplied, unencrypted password it passes via the *sPassword* parameter contains no uppercase characters. If *sPassword* contains any uppercase characters, `PwdVerify()` always returns `false`.

To determine whether you must make the user-supplied, unencrypted string lowercase before calling `PwdVerify()`, you must know the version of Palm OS on the user's device. During HotSync operations, call [`SyncGetHHOSVersion\(\)`](#) to get the device's major Palm OS version number. To use `PwdVerify()` outside of a HotSync operation, you can store the user's Palm OS version on the desktop using the User Data API—[`UmSetInteger\(\)`](#), for example—and then retrieve it later to determine whether you should convert the user-supplied string to lowercase before calling `PwdVerify()`.

**See Also**

[Um GetUserPassword\(\)](#)

## **Password Library**

*PwdVerify*

---

# Desktop Application Notifier API

---

If you need to create a notifier, it must implement the Desktop Application Notification API described in this chapter. A **notifier** is an optional DLL that developers can implement to notify their desktop applications about synchronization activities. HotSync® Manager calls all registered notifiers before synchronization begins, passing them messages that they act upon—for example, a notifier can tell its desktop application to save and stop modifying data. HotSync Manager calls notifiers again after synchronization so that they can, in turn, notify their desktop applications whether the synchronization operation completed successfully.

The Desktop Application Notification functions are declared in `HSNotify.h`.

The sections in this chapter are:

|                                                                  |      |
|------------------------------------------------------------------|------|
| <a href="#">Desktop Application Notifier Constants</a> . . . . . | 1097 |
| <a href="#">Notifier-Defined API Functions</a> . . . . .         | 1099 |

## Desktop Application Notifier Constants

This section describes the following group of related preprocessor constants for this API: [Notification Messages](#).

## Notification Messages

**Purpose** Define messages that HotSync Manager can pass to a notifier when it calls a notifier's [HS\\_Notify\(\)](#) function during a HotSync operation.

**Declared In** HSNotify.h

**Constants**

```
#define HS_SYNC_FAILURE 3
 HotSync Manager sends this message to indicate that a
 synchronization operation has completed unsuccessfully. Your
 HS_Notify() implementation must return a value of TRUE
 for HotSync Manager to continue.
```

```
#define HS_SYNC_QUERYOK 0
 Before synchronization begins, HotSync Manager sends this
 message asking each notifier whether the synchronization
 operation can start. Your HS_Notify() implementation
 must return a value of TRUE for HotSync Manager to
 continue.
```

```
#define HS_SYNC_STARTED 1
 HotSync Manager sends this message to indicate that a
 synchronization operation has started. Your HS_Notify()
 implementation must return a value of TRUE for HotSync
 Manager to continue.
```

```
#define HS_SYNC_SUCCESS 2
 HotSync Manager sends this message to indicate that a
 synchronization operation has completed successfully. Your
 HS_Notify() implementation must return a value of TRUE
 for HotSync Manager to continue.
```

```
#define HS_USER_UPDATE 4
 HotSync Manager sends this message to indicate that the
 user data file has changed. Your HS_Notify()
 implementation must return a value of TRUE for HotSync
 Manager to continue.
```

**Compatibility** HotSync Manager version: All.  
Palm OS version: All.

**See Also** [HS\\_Notify\(\)](#)

## Notifier-Defined API Functions

This section describes the following functions that a notifier implements.

---

|                                             |                                                                                                                       |
|---------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| <a href="#"><u>GetNotifierVersion()</u></a> | Returns a notifier-defined version number.                                                                            |
| <a href="#"><u>HS_Notify()</u></a>          | Receives notification messages from HotSync Manager and, upon return, indicates whether HotSync Manager can continue. |

---

## Desktop Application Notifier API

### *GetNotifierVersion*

---

## GetNotifierVersion Function

|                      |                                                                                                                                                                                                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>       | Returns a notifier-defined version number.                                                                                                                                                                                                                                          |
| <b>Declared In</b>   | HSNotify.h                                                                                                                                                                                                                                                                          |
| <b>Prototype</b>     | <code>DWORD GetNotifierVersion ()</code><br><code>typedef DWORD (*PGETNOTIFIERVERSIONFUNC) ()</code>                                                                                                                                                                                |
| <b>Parameters</b>    | None.                                                                                                                                                                                                                                                                               |
| <b>Returns</b>       | Return the version number of the notifier. This value is defined by the notifier.                                                                                                                                                                                                   |
| <b>Comments</b>      | Implementing <code>GetNotifierVersion()</code> is optional. HotSync Manager does not define what this function should return nor does it call this function. However, it may be useful for your desktop application to call this function to identify the version of your notifier. |
| <b>Compatibility</b> | HotSync Manager version: All.<br>Palm OS version: All.                                                                                                                                                                                                                              |

## **HS\_Notify Function**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Purpose</b>     | Receives notification messages from HotSync Manager and, upon return, indicates whether HotSync Manager can continue.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Declared In</b> | <code>HSNotify.h</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Prototype</b>   | <pre>BOOL HS_Notify (int nCode, int nUserId) typedef BOOL (*PHSNOTIFYFUNC) (int nCode,                                int nUserId)</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Parameters</b>  | <p>→ <i>nCode</i><br/>The message from HotSync Manager, which is one of the constants defined in “<a href="#">Notification Messages</a>” on page 1098.</p> <p>→ <i>nUserId</i><br/>The ID of the user who is synchronizing.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Returns</b>     | If your notifier wants the HotSync operation to start, return TRUE.<br>If your notifier does not want the HotSync operation to start, return FALSE.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Comments</b>    | A notifier must implement this function. HotSync Manager calls a notifier’s <code>HS_Notify()</code> function to pass it the ID of the user ( <i>nUserID</i> ) who started the HotSync operation and basic status messages ( <i>nCode</i> ). Your implementation must return a value of TRUE, otherwise HotSync Manager aborts the synchronization operation. What your notifier does beyond these minimum requirements is up to you. A typical notifier queries its desktop application to determine whether it is ready for its conduit to synchronize desktop data. Your notifier can return FALSE to halt the entire synchronization operation if, for example, your desktop application is currently accessing the data to be synchronized. |

## **Desktop Application Notifier API**

*HS\_Notify*

---



# Part IV Appendices

This part provides the following appendixes:

|                                                       |      |
|-------------------------------------------------------|------|
| <a href="#"><u>Revision History</u></a>               | 1105 |
| <a href="#"><u>Private and Obsolete Functions</u></a> | 1125 |



# A

## Revision History

---

This appendix lists the C/C++ Sync Suite functions and other features that have been added in each version of the CDK since version 4.02. Before you use any new functions or features, you must first check to ensure that they are implemented in the DLL installed on the end user's desktop computer. Most APIs include a function that retrieves the API's version number, which your application should check before calling functions that are not available in all versions. For those APIs that do not have a "get version" function, you can check the version of the HotSync Manager executable.

The sections of this appendix describe changes in these versions of the CDK:

|                                                                  |      |
|------------------------------------------------------------------|------|
| <a href="#">Changes in C/C++ Sync Suite 6.0.1</a> . . . . .      | 1106 |
| <a href="#">Changes in C/C++ Sync Suite 6.0a</a> . . . . .       | 1109 |
| <a href="#">Changes in C/C++ Sync Suite 6.0</a> . . . . .        | 1110 |
| <a href="#">Changes in C/C++ Sync Suite 4.03</a> . . . . .       | 1118 |
| <a href="#">Changes in C/C++ Sync Suite 4.02/4.02a</a> . . . . . | 1119 |

For a summary of the enhancements and new features available in the last few releases of the CDK, see [Chapter 1, “What’s New in the Palm OS CDK,”](#) on page 1 in a *Introduction to Conduit Development*.

## Revision History

*Changes in C/C++ Sync Suite 6.0.1*

---

# Changes in C/C++ Sync Suite 6.0.1

This section lists the C/C++ Sync Suite APIs and other features that changed in version 6.0.1 of the CDK:

|                                                                                        |      |
|----------------------------------------------------------------------------------------|------|
| <a href="#">Sync Manager API, Version 2.5 Changes</a> . . . . .                        | 1106 |
| <a href="#">VFS Manager API, Version 1.1 Changes</a> . . . . .                         | 1107 |
| <a href="#">File Link API Removal from HotSync Manager 6.0.1</a> .                     | 1107 |
| <a href="#">HotSync Log API Changes for HotSync Manager 6.0.1</a> .                    | 1108 |
| <a href="#">User Data API, Version 4.2 Changes</a> . . . . .                           | 1108 |
| <a href="#">Palm OS Common Language API Changes in HotSync Manager 6.0.1</a> . . . . . | 1108 |
| <a href="#">Conduit API Changes in HotSync Manager 6.0.1</a> . . .                     | 1109 |
| <a href="#">HotSync Manager API Changes in HotSync Manager 6.0.1</a> .                 |      |
|                                                                                        | 1109 |

## Sync Manager API, Version 2.5 Changes

Version 2.5 of the Sync Manager API includes these changes:

- Added [SyncBackupDatabase\(\)](#).
- Added [SyncCallDeviceApplication\(\)](#) and [CCallApplicationParams](#).
- Moved [SyncCallRemoteModule\(\)](#) and [CCallModuleParams](#) from SyncCommon.h to SyncMgr.h. This function works only for Palm OS® Garnet or earlier handhelds.
- Removed [SyncDmReadAppPreference\(\)](#) and [SyncDmWriteAppPreference\(\)](#). These functions were designed to read/write **application preferences** whose size is not limited to 64 KB. Palm OS Cobalt, version 6.0.1, supports only preferences smaller than 64 KB, because the preferences are stored in [classic databases](#). Therefore you must use the classic Sync Manager calls [SyncReadAppPreference\(\)](#) and [SyncWriteAppPreference\(\)](#) instead. If your application needs to store preferences larger than 64 KB, then you must use another means—for example store them in your own

database rather than the system's application preferences database.

- A problem is fixed as described in [SyncReadPositionXMap\(\)](#), “[Sync Manager Version 2.5](#)” on page 362.
- Added #define dmHdrAttrClassic and SYNCAPI\_VER\_MINOR\_5.

## VFS Manager API, Version 1.1 Changes

Version 1.1 of the VFS Manager API includes these changes:

- Added [VFSImportDatabaseFromFileEx\(\)](#) and [VFSExportDatabaseToFileEx\(\)](#) to enable conduits to export/import a database image to/from an expansion card on handhelds running Palm OS Cobalt, versions 6.0.1 or later. These functions deprecate the existing functions ([VFSExportDatabaseToFile\(\)](#) and [VFSImportDatabaseFromFile\(\)](#) for use with Palm OS Cobalt handhelds).
- Added #define VFSAPI\_VER\_MINOR\_1.

## File Link API Removal from HotSync Manager 6.0.1

Because the file link feature has been removed from HotSync Manager 6.0.1, the `Subscribe.h` header file, which declared the File Link API, has been removed from CDK 6.0.1.

## Revision History

*Changes in C/C++ Sync Suite 6.0.1*

---

## HotSync Log API Changes for HotSync Manager 6.0.1

The following values have been added to the [Activity](#) enum:

- slSyncSessionStart
- slSyncSessionEnd
- slSyncSessionCancelled
- slError
- slRecommendation
- slHTMLText

Beginning with version 6.0.1 of HotSync Manager, the HotSync log is stored as an HTML file, rather than an ASCII text file. See “[Adding Messages to the HotSync Log](#)” on page 46.

Because the file link feature has been removed from HotSync Manager 6.0.1, the following Activity enum values have been deprecated: slDoubleModifySubsc, slFileLinkCompleted, and slFileLinkDeleted.

## User Data API, Version 4.2 Changes

Version 4.2 of the User Data API includes these changes:

- Added the #defines in “[Palm OS Version Section and Key Names](#)” on page 1013 to enable an application to read the Palm OS version of a user’s handheld. HotSync Manager writes the version to the user data store during each HotSync operation.
- Added [UmIsUserNameValid\(\)](#).
- Added error codes ERR\_UM\_NO\_DEFAULT\_USER and ERR\_UM\_MULTIPLE\_USERS\_EXISTS for future use.

## Palm OS Common Language API Changes in HotSync Manager 6.0.1

In HotSync Manager version 6.0.1, the value for #define LANGUAGE\_TCHINESE was changed from 0x0404 to 0x4000 in LANG\_DLL.h.

## **Conduit API Changes in HotSync Manager 6.0.1**

Because the file link feature has been removed from HotSync Manager 6.0.1, the following conduit entry points have been removed from CondAPI.h:

- ImportData()
- ConfigureSubscription()
- SubscriptionSupported()
- UpdateTables()

The CONDERR\_SUBSCRIBE\_FAILED error code has been deprecated.

## **HotSync Manager API Changes in HotSync Manager 6.0.1**

Because the file link feature has been removed from HotSync Manager 6.0.1, the [HsDisplayFileLink\(\)](#) function has been deprecated.

# **Changes in C/C++ Sync Suite 6.0a**

Version 2.4 of the Sync Manager API includes these changes:

- Moved [SyncCallRemoteModule\(\)](#) and [CCallModuleParams](#) from SyncCommon.h to SyncMgr.h, because this function cannot distinguish between applications that are classic vs. extended resource databases. Therefore call this function only when synchronizing with a Palm OS Garnet or earlier handheld. For Palm OS Cobalt handhelds, call [SyncCallDeviceApplication\(\)](#) instead.

## Revision History

*Changes in C/C++ Sync Suite 6.0*

---

# Changes in C/C++ Sync Suite 6.0

This section lists the C/C++ Sync Suite APIs and other features that changed in version 6.0 of the CDK:

|                                                                 |      |
|-----------------------------------------------------------------|------|
| <a href="#">Sync Manager API, Version 2.4 Changes</a>           | 1110 |
| <a href="#">Conduit Entry Point API Changes</a>                 | 1111 |
| <a href="#">Conduit Manager API, Version 3 Changes</a>          | 1111 |
| <a href="#">Install Conduit Manager API, Version 3 Changes</a>  | 1114 |
| <a href="#">Notifier Install Manager API, Version 3 Changes</a> | 1115 |
| <a href="#">HotSync Log API Changes</a>                         | 1115 |
| <a href="#">HotSync Manager API, Version 2 Changes</a>          | 1116 |
| <a href="#">VFS Manager API, Version 1.0 Changes</a>            | 1116 |
| <a href="#">User Data API, Version 4.1 Changes</a>              | 1116 |
| <a href="#">Install Aide API, Version 4.1 Changes</a>           | 1117 |
| <a href="#">File Linking API Changes</a>                        | 1117 |

## Sync Manager API, Version 2.4 Changes

Version 2.4 of the Sync Manager API includes these changes:

- Added the [Schema Sync Manager API](#): SyncDb...().
- Added the [Extended Sync Manager API](#): SyncDm...().
- Reorganized Sync\*.h header files as described in [Part I, “Sync Manager API.”](#)
- Added the following utility functions to SyncCommon.h described in [Chapter 5, “Common Sync Manager API,”](#) on page 383:
  - [SyncBackupSecurityData\(\)](#)
  - [SyncGenerateBackupFileName\(\)](#)
  - [SyncGetDesktopTrustStatus\(\)](#)
  - [SyncInstallAndBackupDatabase\(\)](#)
  - [SyncInstallDatabase\(\)](#)
  - [SyncIsDatabaseBackupNeeded\(\)](#)
  - [SyncRestoreSecurityData\(\)](#)

- [SyncIsDatabaseBackupNeeded\(\)](#)
- A problem is fixed as described in [SyncReadPositionXMap\(\)](#), “[Sync Manager Version 2.4](#)” on page 362.
- [SyncMaxRemoteRecSize\(\)](#) returns 65,505 as the maximum record size for classic databases regardless of the version of Palm OS on the handheld.

## Conduit Entry Point API Changes

The version of the Conduit Entry Point API that conduits implement to work with HotSync Manager 6.0 and later includes these changes:

- Added the [RegistrationInfoType](#) structure.
- Added the following [ConduitInfoEnum](#) values:
  - eRegistrationInfo
  - eDoNotDisplayInCustomDialog
  - eRunAlways
  - eDoNotDisplayProgress
- Changed HotSync Manager version 6.0 so that it does not need to check a conduit’s MFC version. Therefore HotSync Manager never passes in the eMfcVersion value via the *infoType* parameter of [GetConduitInfo\(\)](#).

These API elements are defined in [Chapter 6, “Conduit Entry Point API,”](#) on page 495.

## Conduit Manager API, Version 3 Changes

Version 3 of the Conduit Manager API includes these changes:

- Added the ability to register different conduits for different Windows users on the same system.
- Added the ability to register a C API-based conduit by copying the DLL to a special folder, where HotSync Manager discovers it and invokes it as with conventionally registered conduits.

## Revision History

Changes in C/C++ Sync Suite 6.0

---

- Deprecated several conduit configuration entries that HotSync Manager does not use the APIs that get and set them.
- Added a new, fixed-size structure, [CmConduitType2](#), to specify and receive only the required and most commonly used conduit configuration entries. Also added functions that use this structure.
- Removed several #define constants that had no purpose: BACKUP\_CONDUIT, MAX\_SPECIAL, INVALID\_CREATORID, INVALID\_INTEGRATE, and INVALID\_PRIORITY.
- [CmGetConduitCreatorID\(\)](#) correctly modifies the *piSize* parameter upon return to the actual or required size of the creator ID string. In earlier versions of Conduit Manager, it did not.
- Added the following Conduit Manager functions:
  - [CmGetDiscoveryInfoByIndex\(\)](#)
  - [CmGetSystemBackupConduit\(\)](#)
  - [CmGetSystemConduitByCreator\(\)](#)
  - [CmGetSystemConduitByIndex\(\)](#)
  - [CmGetSystemConduitCount\(\)](#)
  - [CmGetSystemConduitCreatorID\(\)](#)
  - [CmGetSystemCreatorDirectory\(\)](#)
  - [CmGetSystemCreatorFile\(\)](#)
  - [CmGetSystemCreatorIDLList\(\)](#)
  - [CmGetSystemCreatorName\(\)](#)
  - [CmGetSystemCreatorPriority\(\)](#)
  - [CmGetSystemCreatorRemote\(\)](#)
  - [CmGetSystemCreatorTitle\(\)](#)
  - [CmGetSystemCreatorValueDword\(\)](#)
  - [CmGetSystemCreatorValueString\(\)](#)
  - [CmGetSystemDiscoveryInfoByIndex\(\)](#)
  - [CmGetSystemHotSyncExecPath\(\)](#)
  - [CmInstallSystemConduitByStruct\(\)](#)

- [CmInstallSystemCreator\(\)](#)
- [CmIsCurrentUserAdmin\(\)](#)
- [CmRemoveSystemConduitByCreatorID\(\)](#)
- [CmRemoveSystemConduitByIndex\(\)](#)
- [CmSetSystemBackupConduit\(\)](#)
- [CmSetSystemCreatorDirectory\(\)](#)
- [CmSetSystemCreatorFile\(\)](#)
- [CmSetSystemCreatorName\(\)](#)
- [CmSetSystemCreatorPriority\(\)](#)
- [CmSetSystemCreatorRemote\(\)](#)
- [CmSetSystemCreatorTitle\(\)](#)
- [CmSetSystemCreatorValueDword\(\)](#)
- [CmSetSystemCreatorValueString\(\)](#)
- [FmDisableCurrentUserConduitByIndex\(\)](#)
- [FmDisableCurrentUserConduitByPath\(\)](#)
- [FmDisableSystemConduitByIndex\(\)](#)
- [FmDisableSystemConduitByPath\(\)](#)
- [FmEnableCurrentUserConduitByPath\(\)](#)
- [FmEnableSystemConduitByPath\(\)](#)
- [FmGetCurrentUserConduitFolder\(\)](#)
- [FmGetCurrentUserDisabledConduitFolder\(\)](#)
- [FmGetSystemConduitByIndex\(\)](#)
- [FmGetSystemConduitCount\(\)](#)
- [FmGetSystemConduitFolder\(\)](#)
- [FmGetSystemDisabledConduitFolder\(\)](#)

## Revision History

*Changes in C/C++ Sync Suite 6.0*

---

## Install Conduit Manager API, Version 3 Changes

Version 3 of the Install Conduit Manager API includes these changes:

- Added the ability to register different install conduits for different Windows users on the same system.
- Added the following functions to enable installers to register install conduits for the system:
  - [ImGetSystemDirectory\(\)](#)
  - [ImGetSystemDWord\(\)](#)
  - [ImGetSystemExtension\(\)](#)
  - [ImGetSystemMask\(\)](#)
  - [ImGetSystemModule\(\)](#)
  - [ImGetSystemName\(\)](#)
  - [ImGetSystemString\(\)](#)
  - [ImRegisterSystem\(\)](#)
  - [ImSetSystemDirectory\(\)](#)
  - [ImSetSystemDWord\(\)](#)
  - [ImSetSystemExtension\(\)](#)
  - [ImSetSystemMask\(\)](#)
  - [ImSetSystemModule\(\)](#)
  - [ImSetSystemName\(\)](#)
  - [ImSetSystemString\(\)](#)
  - [ImUnregisterSystemID\(\)](#)

## Notifier Install Manager API, Version 3 Changes

Version 3 of the Notifier Install Manager API includes these changes:

- Added the ability to register different notifiers for different Windows users on the same system.
- Added the following functions to enable installers to register notifiers for the system:
  - [NmFindSystem\(\)](#)
  - [NmGetSystemByIndex\(\)](#)
  - [NmGetSystemCount\(\)](#)
  - [NmRegisterSystem\(\)](#)
  - [NmRenameSystemByIndex\(\)](#)
  - [NmUnregisterSystem\(\)](#)

## HotSync Log API Changes

The version of the HotSync Log API that ships with Sync Manager versions 2.4 and later include these changes:

- Added the `s1SyncDidNothing` value to the [Activity](#) enum.
- Made the following HotSync Log API functions private and no longer document them in “[HotSync Log Functions](#)” on page 534:
  - `LogBuildRemoteLog()`
  - `LogCloseLog()`
  - `LogGetWorkFileName()`
  - `LogInit()`
  - `LogSaveLog()`
  - `LogUnInit()`

Only [LogAddEntry\(\)](#), [LogAddFormattedEntry\(\)](#), and [LogTestCounters\(\)](#) are supported for conduits to use.

## Revision History

Changes in C/C++ Sync Suite 6.0

---

## HotSync Manager API, Version 2 Changes

Version 2 of the HotSync Manager API includes these changes:

- Added the HSFLAG\_INSPECT\_CONDUIT constant so that [HsSetAppStatus\(\)](#) can start Conduit Inspector when it starts HotSync Manager. To do so, the caller must pass in the this constant in the *dwStartFlags* parameter. See “[Start Options](#)” on page 932.
- Added the HS\_API\_VERSION\_2 constant as the value that [HsGetApiVersion\(\)](#) passes back. This constant is defined in “[HotSync Manager API Versions](#)” on page 929.

## VFS Manager API, Version 1.0 Changes

Version 1.0 of the VFS Manager API includes these changes:

- Added the vfsVolumeAttrHidden value to the VFSMgr.h file. It is described in “[Volume Attributes](#)” on page 578.
- Removed the vfsMountClass\_POSE constant from VFSMgr.h, because it cannot be used from the desktop VFS Manager.
- Added the “[Volume Format/Mount Flags](#)” on page 580, which can be passed in the *byMountFlags* parameter when calling [VFSVolumeFormat\(\)](#).

Note that the VFS Manager API version number has not changed; [VFSGetAPIVersion\(\)](#) returns version 1.0 as it did in CDK 4.03.

## User Data API, Version 4.1 Changes

Version 4.1 of the User Data API includes these changes:

- Added the UM\_LIB\_VER\_MAJOR and UM\_LIB\_VER\_MINOR constants as the values that [UmGetLibVersion\(\)](#) passes back. These constants are defined in “[Version of the User Data API](#)” on page 1014.

Note that the User Data API version number has not changed; UmGetLibVersion() returns version 4.1 as it did in CDK 4.03.

## Install Aide API, Version 4.1 Changes

Version 4.1 of the Install Aide API includes these changes:

- Added the [PltRemoveInstallRegistry\(\)](#) function to remove the configuration entries of the install conduits set by [PltSetInstallRegistry\(\)](#).
- Added the INSTAIDE\_LIB\_VER\_MAJOR and INSTAIDE\_LIB\_VER\_MINOR constants as the values that [PlmGetLibVersion\(\)](#) passes back. These new constants are defined in “[Version of the Install Aide API](#)” on page 957.

## File Linking API Changes

In HotSync Manager version 6.0, file linking is deprecated. Conduits that implement file linking with any database type (schema, extended, or classic) continue to function normally, but the limitations on categories remain in effect: only 16 categories with names limited to 15 characters. Note that in certain upgrade scenarios, records in a file-link category are duplicated.

If you are developing with Visual C++ .NET, you must link to Subs50.lib; if you are using Visual C++ version 6, link to Subs30.lib.

## Palm OS Common Language API Changes

In HotSync Manager version 6.0, #define LANGUAGE\_SCHINESE was added in LANG\_DLL.h.

## Revision History

*Changes in C/C++ Sync Suite 4.03*

---

# Changes in C/C++ Sync Suite 4.03

This section lists the C/C++ Sync Suite APIs and other features that changed in version 4.03 of the CDK:

- [Sync Manager API Changes](#)
- [HotSync Manager API Changes](#)
- [User Data API, Version 4.1 Changes](#)

## Sync Manager API Changes

Additional values for the eDbFlags enum have been defined as described in “[eDbFlags](#)” on page 416. The new values are:

- eStream
- eHidden
- eLaunchableData
- eRecyclable
- eBundle

## HotSync Manager API Changes

In the HSAPI.h file, the HSConnectionType enum described in “[HSConnectionType](#)” on page 930 now includes values for indicating an infrared (CtIR) and a USB (CtUSB) connection type.

## User Data API, Version 4.1 Changes

Version 4.1 of the User Data API addresses a problem in the [UmGetUserPassword\(\)](#) function. If the encrypted user password happens to contain a NULL character, the function returned only the characters of the encrypted password up to the NULL character. In version 4.1, this function returns the entire encrypted password as expected.

## Changes in C/C++ Sync Suite 4.02/4.02a

This section lists the C/C++ Sync Suite APIs and other features that changed in version 4.02/4.02a of the CDK:

- [Expansion Manager and VFS Manager API, Version 1.0 Changes](#)
- [Sync Manager API, Version 2.3 Changes](#)
- [User Data API, Version 4.0 Changes](#)
- [Install Aide API, Version 4.0 Changes](#)
- [Conduit Manager, Version 2 Changes](#)
- [Password Library Changes](#)
- [HotSync Manager API Changes](#)

### Expansion Manager and VFS Manager API, Version 1.0 Changes

Version 1.0 is the first release of the Expansion Manager and Virtual File System Manager (VFS) APIs. Use the [VFSGetAPIVersion\(\)](#) function to get the version number of these desktop APIs.

See [Chapter 9, “Expansion Manager API,”](#) on page 547 and [Chapter 10, “Virtual File System Manager API,”](#) on page 561 for details on these APIs.

### Sync Manager API, Version 2.3 Changes

Version 2.3 of the Sync Manager changed in two ways described in these sections:

- [Functions](#)
- [Change to Support More than 32 Databases](#)

#### Functions

Version 2.3 of the Sync Manager adds the `SyncLoopBackTest()` function, a private function for test purposes that conduit developers must not call.

## Revision History

Changes in C/C++ Sync Suite 4.02/4.02a

---

### Change to Support More than 32 Databases

Version 2.3 of the Sync Manager fixes a problem that crashes the Sync Manager if your conduit is associated with more than 32 databases with the same creator ID. In the [CSyncProperties](#) class, the `m_RemoteName` array is erroneously limited to `SYNC_DB_NAMELEN` entries. Therefore previous versions of the Sync Manager may crash if your conduit is associated with more than 32 databases with the same creator ID on the handheld. To maintain backward compatibility, this limit remains, but Sync Manager now prevents `m_RemoteName` from overflowing and corrupting subsequent data in `CSyncProperties` if the number of databases is more than 32. However, if your conduit is associated with more than 32 databases, get a list of their names from `m_RemoteDbList` instead of `m_RemoteName`.

## User Data API, Version 4.0 Changes

Version 4.0 of the User Data API adds support for expansion slots on Palm Powered™ handhelds. See “[User Data API Versions](#)” on page 1009 for how to distinguish versions of the User Data API.

### Functions

This version adds the following functions to the User Data API:

[UmGetLibVersion\(\)](#)      [UmSlotGetInstallDirectory\(\)](#)  
[UmSlotGetDisplayName\(\)](#)    [UmSlotGetMediaType\(\)](#)  
[UmSlotGetExpMgrVersion\(\)](#) [UmSlotGetSlotCount\(\)](#)  
[UmSlotGetInfo\(\)](#)

In version 4.0, [UmGetUserID\(\)](#), if successful, returns 0 and passes back the user ID in `pdwUserID`; if unsuccessful, it returns a negative error code. In versions *earlier* than 4.0, `UmGetUserID`, if successful, returned the user ID (a value that could be either positive or negative before it is cast) and passed the same value back in `pdwUserID`; if unsuccessful, it returned 0 if no user ID is found or and a negative value on error. Therefore a negative return value may or may not have indicated an error. Version 4.0 fixes this problem.

## Error Codes

Version 4.0 adds the following error codes:

[ERR\\_UM\\_DEV\\_CFG\\_DATA\\_NOT\\_AVAILABLE](#)

## Install Aide API, Version 4.0 Changes

This version of the Install Aide API adds support for expansion slots on Palm Powered handhelds.

## Functions

This version adds the following functions to the Install Aide API:

|                                                       |                                                         |
|-------------------------------------------------------|---------------------------------------------------------|
| <a href="#"><u>PlmGetLibVersion()</u></a>             | <a href="#"><u>PlmSlotInstallFile()</u></a>             |
| <a href="#"><u>PlmMoveInstallFileToHandheld()</u></a> | <a href="#"><u>PlmSlotMoveInstallFile()</u></a>         |
| <a href="#"><u>PlmMoveInstallFileToSlot()</u></a>     | <a href="#"><u>PlmSlotRemoveInstallFile()</u></a>       |
| <a href="#"><u>PlmSlotGetFileCount()</u></a>          | <a href="#"><u>PltGetInstallFileFilterForUser()</u></a> |
| <a href="#"><u>PlmSlotGetFileInfo()</u></a>           |                                                         |

In version 4.0, [PltRemoveInstallFile\(\)](#), if unsuccessful, returns a `ERR_PALM_FILE_DELETE_FAILED` error code. In earlier versions, this function did not return an error message if the specified file did not exist.

With version 4.02 of the C/C++ Sync Suite, all functions available in `InstAide.dll` are now declared in `InstAide.h` rather than `InstAppd.h`. Update your source to include `InstAide.h` instead.

## Error Codes

This version adds the following error codes:

[ERR\\_PALM\\_FILE\\_DELETE\\_FAILED](#)

[ERR\\_PALM\\_FILE\\_MOVE\\_FAILED](#)

## Revision History

*Changes in C/C++ Sync Suite 4.02/4.02a*

---

## Conduit Manager, Version 2 Changes

This version of the Conduit Manager includes Install Conduit Manager and Notifier Install Manager functions. Functions for all three managers are available in CondMgr.dll.

With version 4.02 of the C/C++ Sync Suite, all functions available in CondMgr.dll are now declared in CondMgr.h rather than CondMgre.h. Update your source to include CondMgr.h instead.

### New Functions

This version adds the following Install Conduit Manager and Notifier Install Manager functions:

|                                  |                                   |
|----------------------------------|-----------------------------------|
| <a href="#">ImGetDirectory()</a> | <a href="#">ImSetMask()</a>       |
| <a href="#">ImGetDWord()</a>     | <a href="#">ImSetModule()</a>     |
| <a href="#">ImGetExtension()</a> | <a href="#">ImSetName()</a>       |
| <a href="#">ImGetMask()</a>      | <a href="#">ImSetString()</a>     |
| <a href="#">ImGetModule()</a>    | <a href="#">ImUnregisterID()</a>  |
| <a href="#">ImGetName()</a>      | <a href="#">NmFind()</a>          |
| <a href="#">ImGetString()</a>    | <a href="#">NmGetByIndex()</a>    |
| <a href="#">ImRegister()</a>     | <a href="#">NmGetCount()</a>      |
| <a href="#">ImRegisterID()</a>   | <a href="#">NmRegister()</a>      |
| <a href="#">ImSetDirectory()</a> | <a href="#">NmRenameByIndex()</a> |
| <a href="#">ImSetDWord()</a>     | <a href="#">NmUnregister()</a>    |
| <a href="#">ImSetExtension()</a> |                                   |

### Deprecated Functions

Because the Notifier Install Manager introduces functions for handling notifiers, version 2 of the Conduit Manager API deprecates these functions:

- CmSetNotifierDll()
- CmGetNotifierDll()

Though these functions continue to work in version 2 of the Conduit Manager API, they may not in the future, so use the Notifier Install Manager functions instead. See [Chapter 13, “Notifier Install Manager API,”](#) on page 909 for details.

### New Error Codes

This version of Conduit Manager adds the following error codes:

[ERR\\_ALREADY\\_INSTALLED](#)

[ERR\\_CREATORID\\_ALREADY\\_IN\\_USE](#)

[ERR\\_INVALID\\_INSTALL\\_ID](#)

[ERR\\_INVALID\\_PATH](#)

[ERR\\_NOTIFIER\\_NOT\\_FOUND](#)

[ERR\\_STORAGE\\_ACCESS](#)

### Password Library Changes

The password encryption algorithm has been changed in Palm OS® version 4.0. Also in Palm OS 4.0 and later, passwords on the device are case-sensitive; therefore `PwdVerify()` returns `true` only if the user-supplied string is identical, including case, to the stored password. See the description of [PwdVerify\(\)](#) for more information.

### HotSync Manager API Changes

The [HsRefreshConduitInfo\(\)](#) function was added to the HotSync Manager API.

## **Revision History**

*Changes in C/C++ Sync Suite 4.02/4.02a*

---

# Private and Obsolete Functions

---

Some functions visible in the C/C++ Sync Suite or in previous versions of the CDK are not supported for use in your conduit.

The sections in this appendix are:

- [Private Functions](#)
- [Obsolete Functions](#)

## Private Functions

This section lists the private functions embedded in the API DLLs for internal use only. These are private, system functions that you must not use in your conduit. This list is documented here only because you may see these function names in the header files, source code, Dependency Walker, or DLL QuickViews and may not recognize the implications of using them.

---

**IMPORTANT:** Do not use these functions in your code. Doing so can interfere with HotSync® Manager and corrupt all data on the handheld.

---

## **Private and Obsolete Functions**

### *Private Functions*

---

Sync20.dll:

- SyncDeInit
- SyncEndOfSync
- SyncInit
- SyncLoopBackTest
- SyncPreSendCmd
- SyncReadProdCompInfo
- SyncReadNetSyncInfo
- SyncWriteNetSyncInfo
- SyncWriteUserID

PalmCmn.dll:

- PalmAboutBox
- PalmGetResourceVersion
- PsDelete
- PsGetInteger
- PsGetNameByIndex
- PsGetNameCount
- PsGetSectionByIndex
- PsGetSectionCount
- PsGetString
- PsGetVersion
- PsSetInteger
- PsSetString

CondMgr.dll:

- CmGetCreatorRegPath
- Ui\* functions
- Pi\* functions
- ImGetPrerequisites
- ImSetPrerequisites

HSLog20.dll:

- LogBuildRemoteLog()
- LogCloseLog()
- LogGetWorkFileName()
- LogInit()
- LogSaveLog()
- LogUnInit()

VFSAPI.dll:

- VFSInit
- VFSDeinit

## **Private and Obsolete Functions**

### *Obsolete Functions*

---

## **Obsolete Functions**

This section lists the functions that are no longer available in the C/C++ Sync Suite.

| <b>Function Name</b>    | <b>Notes</b>                                                                                                                                                                         |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SyncCallApplication()   | Available in Palm OS® versions earlier than version 2.0. If you call this function on a handheld running version 2.0 or later, your application receives an “illegal request” error. |
| UmSetUserPassword()     | Returns ERR_UM_FUNCTION_NOT_SUPPORTED.                                                                                                                                               |
| CmRemoveConduitByName() | Returns ERR_NO_LONGER_SUPPORTED.                                                                                                                                                     |

# Index

---

## A

Activity 530  
ARCHIVE\_BIT 310

## B

BIG\_PATH 429

## C

C/C++ Sync Suite  
    revision history, APIs 1105  
catCategoryNameLength 39  
categories  
    definition in a schema database 10  
    matching in a schema database 30  
CategoryID 8  
catIDAll 30  
catIDUnfiled 30  
CCallApplicationParams 385  
CCallModuleParams 291  
CCardInfo 387  
CDBCreateDB 293  
CDbGenInfo 295  
CDBList 389  
CDBListPtr 389  
CDK  
    documentation xxvi  
CfgConduit() 511  
CFGCONDUITINFO\_VERSION\_1 509  
CfgConduitInfoType 497  
charEncodingAscii 31  
charEncodingCP1252 31  
charEncodingCP932 31  
charEncodingISO8859\_1 31  
charEncodingPalmLatin 31  
charEncodingPalmSJIS 31  
charEncodingShiftJIS 31  
CharEncodingType 8  
charEncodingUnknown 31  
charEncodingUTF8 31  
classic databases  
    Sync Manager  
        C API 289

CM\_CONDUIT\_BUFFER\_OFFSET 664  
CM\_CREATOR\_ID\_SIZE 664  
CM\_CREATORLIST\_ITEM\_TYPE 659  
CM\_CREATORLIST\_TYPE 659  
CM\_INITIAL\_LIB\_VERSION 663  
CM\_MIN\_CONDUITTYPE\_SIZE 664  
CM\_UPDATE\_1 663  
CM\_UPDATE\_2 663  
CmConduitType 653  
CmConduitType2 656  
CmConvertCreatorIDToString() 676  
CmConvertStringToCreatorID() 677  
CmDiscoveryInfoType 658  
CmGetBackupConduit() 678  
CmGetComPort() 680  
CmGetConduitByCreator() 682  
CmGetConduitByIndex() 684  
CmGetConduitCount() 685  
CmGetConduitCreatorID() 686  
CmGetCorePath() 688  
CmGetCreatorArgument() 690  
CmGetCreatorDirectory() 692  
CmGetCreatorFile() 694  
CmGetCreatorIDLList() 696  
CmGetCreatorInfo() 698  
CmGetCreatorIntegrate() 700  
CmGetCreatorModule() 701  
CmGetCreatorName() 703  
CmGetCreatorPriority() 705  
CmGetCreatorRegPath() 1127  
CmGetCreatorRemote() 707  
CmGetCreatorTitle() 709  
CmGetCreatorType() 711  
CmGetCreatorUser() 712  
CmGetCreatorValueDword() 714  
CmGetCreatorValueString() 716  
CmGetDiscoveryInfoByIndex() 718  
CmGetHotSyncExecPath() 719  
CmGetLibVersion() 721  
CmGetNotifierDll() 722  
CmGetPCIIdentifier() 724  
CmGetSystemBackupConduit() 725  
CmGetSystemConduitByCreator() 727

---

CmGetSystemConduitByIndex() 728  
CmGetSystemConduitCount() 729  
CmGetSystemConduitCreatorID() 730  
CmGetSystemCreatorDirectory() 732  
CmGetSystemCreatorFile() 734  
CmGetSystemCreatorIDLList() 736  
CmGetSystemCreatorName() 738  
CmGetSystemCreatorPriority() 740  
CmGetSystemCreatorRemote() 742  
CmGetSystemCreatorTitle() 744  
CmGetSystemCreatorValueDword() 746  
CmGetSystemCreatorValueString() 748  
CmGetSystemDiscoveryInfoByIndex() 750  
CmGetSystemHotSyncExecPath() 751  
CmInstallConduit() 754  
CmInstallConduitByStruct() 756  
CmInstallCreator() 758  
CmInstallSystemConduitByStruct() 760  
CmInstallSystemCreator() 762  
CmIsCurrentUserAdmin() 764  
CmRemoveConduitByCreatorID() 765  
CmRemoveConduitByIndex() 767  
CmRemoveConduitByName() 1128  
CmRemoveSystemConduitByCreatorID() 769  
CmRemoveSystemConduitByIndex() 771  
CmRestoreHotSyncSettings() 773  
CmSetBackupConduit() 775  
CmSetComPort() 776  
CmSetCorePath() 777  
CmSetCreatorArgument() 778  
CmSetCreatorDirectory() 779  
CmSetCreatorFile() 781  
CmSetCreatorInfo() 783  
CmSetCreatorIntegrate() 784  
CmSetCreatorModule() 785  
CmSetCreatorName() 786  
CmSetCreatorPriority() 788  
CmSetCreatorRemote() 790  
CmSetCreatorTitle() 792  
CmSetCreatorUser() 794  
CmSetCreatorValueDword() 795  
CmSetCreatorValueString() 797  
CmSetHotSyncExecPath() 799  
CmSetNotifierDll() 800  
CmSetPCIIdentifier() 801  
CmSetSystemBackupConduit() 802  
CmSetSystemCreatorDirectory() 803  
CmSetSystemCreatorFile() 805  
CmSetSystemCreatorName() 807  
CmSetSystemCreatorPriority() 809  
CmSetSystemCreatorRemote() 811  
CmSetSystemCreatorTitle() 813  
CmSetSystemCreatorValueDword() 815  
CmSetSystemCreatorValueString() 817  
ColumnID 9  
common Sync Manager API 383  
compatibility  
    APIs, C/C++ Sync Suite 1105  
COND\_ERR\_CLASS 486  
CondAPI.h 495  
CONDERR\_ABORT\_DB\_INSTALL 526  
CONDERR\_ADD\_LOCAL\_RECORD 525  
CONDERR\_ADD\_REMOTE\_RECORD 525  
CONDERR\_BAD\_LOCAL\_BACKUP 525  
CONDERR\_BAD\_LOCAL\_TABLES 525  
CONDERR\_BAD\_REMOTE\_TABLES 525  
CONDERR\_BAD\_SYNC\_TYPE 526  
CONDERR\_BUFFER\_TOO\_SMALL 527  
CONDERR\_CHANGE\_REMOTE\_RECORD 525  
CONDERR\_CONDUIT\_RESOURCE\_FAILURE  
    527  
CONDERR\_CONVERT\_FROM\_REMOTE\_REC  
    526  
CONDERR\_CONVERT\_TO\_LOCAL\_CATS 526  
CONDERR\_CONVERT\_TO\_REMOTE\_CATS 525  
CONDERR\_CONVERT\_TO\_REMOTE\_REC 526  
CONDERR\_DATE\_MOVED 526  
CONDERR\_FIRST 524  
CONDERR\_INVALID\_ARGSSIZE 527  
CONDERR\_INVALID\_ARGSSIZE\_PTR 527  
CONDERR\_INVALID\_BUFFER\_SIZE 527  
CONDERR\_INVALID\_INARGS\_PTR 527  
CONDERR\_INVALID\_INARGS\_STRUCT 527  
CONDERR\_INVALID\_OUTSIZE\_PTR 527  
CONDERR\_INVALID\_PTR 527  
CONDERR\_LOCAL\_MEMORY\_ALLOC\_FAILED  
    525

---

CONDERR\_NO\_LOCAL\_CATEGORIES 524  
CONDERR\_NO\_REMOTE\_CATEGORIES 524  
CONDERR\_NOCLIENTINFO\_AVAILABLE 527  
CONDERR\_NONE 524  
CONDERR\_RAW\_RECORD\_ALLOCATE 525  
CONDERR\_REMOTE\_CHANGES\_NOT\_SENT 525  
CONDERR\_REMOTE\_RECS\_NOT\_PURGED 526  
CONDERR\_SAVE\_REMOTE\_CATEGORIES 524  
CONDERR\_SUBSCRIBE\_FAILED 526  
CONDERR\_UNSUPPORTED\_CFGCONDUIT\_ENUM 527  
CONDERR\_UNSUPPORTED\_CONDUITINFO\_ENUM 527  
CONDERR\_UNSUPPORTED\_STRUCT\_VERSION 527  
CONDHANDLE 400  
CondMgr.dll 651, 847, 909  
CondMgr.h 651, 847, 909, 1122  
CondMgre.h 1122  
Conduit Manager  
    C API 651  
CONDUIT\_APPLICATION 661  
CONDUIT\_COMPONENT 661  
CONDUIT\_CONDUITS 661  
CONDUIT\_VERSION 664  
ConduitCfgEnum 503  
ConduitInfoEnum 504  
CONDUITREQUESTINFO\_VERSION\_1 509  
ConduitRequestInfoType 499  
conduits  
    entry points  
        C API 495  
ConfigureConduit() 513  
CPositionInfo 297  
CRawPreferenceInfo 299  
CRawRecordInfo 301  
CSyncPreference 391  
CSyncProperties 393  
CSystemInfo 396  
CtIr 930  
CtModemPort 930  
CtNetwork 930  
CtReserved 930  
CtSerialPort 930  
CtUSB 930  
CUserIDInfo 398  
Custom 1012

## D

DB\_NAMELEN 414  
dbAllRecAttrs 40  
dbAllSchemaColAttrs 47  
dbBlob 48  
dbBoolean 48  
DbCategoryDefn 10  
dbChar 48  
dbClassicDatabases 35  
DbColumnData 10  
DbColumnDefn 11  
dbColumnPropertyUpperBound 39  
DbColumnRemove 13  
DbColumnValue 14  
DBDatabaseInfo 401  
dbDatabasesAll 35  
dbDatabasesInRAM 35  
dbDatabasesInROM 35  
dbDataModified 41  
dbDataNoChange 41  
dbDate 48  
dbDateTime 48  
dbDateTimeSecs 48  
dbDeletesPurged 45  
dbDouble 48  
dbExtendedDatabases 35  
dbFastSync 46  
dbFlagExcludeFromSync 411  
dbFlagRamBased 411  
dbFloat 48  
dbInt16 48  
dbInt32 48  
dbInt64 48  
dbInt8 48  
DbMatchAll 30  
DbMatchAny 30  
DbMatchExact 30  
DbMatchModeType 16

---

dbMembershipModified 41  
dbModeReadOnly 37  
dbModeReadWrite 37  
dbModeShowSecret 37  
dbNoChange 46  
dbNoFlag 45  
dbNonSecureDatabases 35  
dbRecAttrArchive 40  
dbRecAttrDelete 40  
dbRecAttrReadOnly 40  
dbRecAttrSecret 40  
dbRecordDatabases 35  
dbResourceDatabases 35  
dbRowAllDataSet 42  
dbRowArchived 41  
dbRowCategoriesDataSet 42  
DbRowChangeFlags 16  
dbRowColumnInfoDataSet 42  
dbRowColumnValuesDataSet 42  
DbRowData 17  
dbRowDeleted 41  
dbRowEmptyDataSet 42  
dbRowInfoDataSet 42  
DbRowResult 20  
dbSchemaColDynamic 47  
dbSchemaColNonSyncable 47  
DbSchemaColumnAttrib 21  
DbSchemaColumnProperty 22  
DbSchemaColumnType 23  
dbSchemaColWritable 47  
dbSchemaDatabases 36  
dbSecureDatabases 36  
DbShareModeType 23  
dbShareNone 38  
dbShareRead 38  
dbShareReadWrite 38  
dbSlowSync 46  
dbStringVector 49  
DbSyncMode 24  
DbSyncType 25  
dbSysOnlyRecAttrs 40  
DbTableDefn 26  
dbTime 49  
dbUInt16 49  
dbUInt32 49  
dbUInt64 49  
dbUInt8 49  
dbVarChar 49  
dbVector 49  
DELETE\_BIT 310  
DEVICE\_INFO\_SECTION\_NAME 1013  
DEVICE\_ROM\_VERSION 1013  
DIRECT\_COM\_PORT 662  
DIRTY\_BIT 310  
dmAllHdrAttrs 407  
dmAllRecAttrs 432  
dmColumnNameLength 39  
dmDBNameLength 431  
dmHdrAttrAppInfoDirty 407  
dmHdrAttrBackup 407  
dmHdrAttrBundle 407  
dmHdrAttrClassic 407  
dmHdrAttrCopyPrevention 407  
dmHdrAttrExtended 408  
dmHdrAttrFixedUp 408  
dmHdrAttrHidden 408  
dmHdrAttrLaunchableData 408  
dmHdrAttrOKToInstallNewer 408  
dmHdrAttrOpen 408  
dmHdrAttrReadOnly 408  
dmHdrAttrRecyclable 408  
dmHdrAttrResDB 408  
dmHdrAttrResetAfterInstall 408  
dmHdrAttrSchema 409  
dmHdrAttrSecure 409  
dmHdrAttrStream 409  
dmModeExclusive 37  
dmModeWrite 37  
dmRecAttrBusy 432  
dmRecAttrDelete 432  
dmRecAttrDirty 432  
dmRecAttrSecret 432  
dmSysOnlyHdrAttrs 409  
dmSysOnlyRecAttrs 432  
dmTableNameLength 39  
DmVaultCreator 39

---

DmVaultType 39  
documentation xxvi  
DoNothing 1012

## E

eAppInfoDirty 416  
eBackup 426  
eBackupDB 416  
eBundle 417  
eCable 415  
eConduitCfgDoNotUse 503  
eConduitInfoDoNotUse 505  
eConduitName 504  
eConfig1 503  
eConnType 415  
eConnTypeDoNotUse 415  
eCopyPrevention 416  
eDbExclusive 419  
eDbFlags 416  
eDbFlagsDoNotUse 418  
eDbOpenModes 419  
eDbOpenModesDoNotUse 419  
eDbRead 419  
eDbShowSecret 419  
eDbWrite 419  
eDefaultAction 504  
eDesktopNotTrusted 421  
eDesktopTrusted 421  
eDesktopTrustNotVerified 421  
eDesktopTrustStatus 421  
eDoNotDisplayInConduitListForUser 504  
eDoNotDisplayProgress 505  
eDoNothing 427  
eExcludeFromSync 414  
eFast 426  
eFirstSync 422  
eFirstSyncDoNotUse 422  
eHH 422  
eHHtoPC 426  
eHidden 417  
eInstall 426  
eLaunchableData 417  
eMfcVersion 504

eMiscDbFlagExcludeFromSync 423  
eMiscDbFlagRamBased 423  
eMiscDbFlagsDoNotUse 423  
eMiscDbListFlags 423  
eModem 414  
eModemConnType 415  
eNeither 422  
eNoPreference 424  
eOkToInstallNewer 416  
eOpenDB 417  
ePCToHH 426  
ePermanentPreference 424  
eProfileInstall 427  
eReadOnly 416  
eRecAttrArchived 425  
eRecAttrBusy 425  
eRecAttrDeleted 425  
eRecAttrDirty 425  
eRecAttrSecret 425  
eRecord 416  
eRecyclable 417  
eRegistrationInfo 504  
eResetAfterInstall 416  
eResource 416  
ERR\_ALREADY\_INSTALLED 844  
ERR\_AMBIGUOUS\_CREATORID 843  
ERR\_BUFFER\_TOO\_SMALL 845  
ERR\_CONDUIT\_MGR 846  
ERR\_CONDUIT\_READ\_ONLY 843  
ERR\_CONFLICTING\_EXTENSION 843  
ERR\_COULD\_NOT\_CREATE\_DIRECTORY 843  
ERR\_CREATORID\_ALREADY\_IN\_USE 846  
ERR\_FOLDER\_NOT\_FOUND 843  
ERR\_IA\_INVALID\_FILE\_TYPE 1004  
ERR\_IA\_INVALID\_USER 1008  
ERR\_IA\_INVALID\_USER\_ID 1005  
ERR\_INDEX\_OUT\_OF\_RANGE 846  
ERR\_INSTALL\_ID\_ALREADY\_IN\_USE 844  
ERR\_INSUFFICIENT\_PRIVILEGES 844  
ERR\_INVALID\_COM\_PORT\_TYPE 844  
ERR\_INVALID\_CONDUIT\_TYPE 845  
ERR\_INVALID\_CREATOR\_ID 845

---

ERR\_INVALID\_HANDLE 845  
ERR\_INVALID\_INSTALL\_ID 844  
ERR\_INVALID\_PATH 844  
ERR\_INVALID\_POINTER 845  
ERR\_NO\_CONDUIT 846  
ERR\_NO\_LONGER\_SUPPORTED 844  
ERR\_NO\_MEMORY 846  
ERR\_NOTIFIER\_NOT\_FOUND 844  
ERR\_PALM\_BAD\_FILENAME 1004  
ERR\_PALM\_FILE\_DELETE\_FAILED 1005  
ERR\_PALM\_FILE\_MOVE\_FAILED 1005  
ERR\_PALM\_NO\_CORE\_PATH 1006  
ERR\_PALM\_NO\_DIRECTORY 1005  
ERR\_PALM\_NO\_USER\_FILE 1005  
ERR\_PALM\_NO\_USERS 1004  
ERR\_PALM\_NOT\_FOUND 1004  
ERR\_PALM\_OTHER\_USERSDAT\_ACCESS\_PROBLEM 1005  
ERR\_PALM\_SEMAPHORE\_ACCESS 1005  
ERR\_PALM\_UNABLE\_TO\_CREATE\_NEW\_FILE 1006  
ERR\_PALM\_USER\_ACCESS 1004  
ERR\_PILOT\_BAD\_FILENAME 1004  
ERR\_PILOT\_BUFSIZE\_TOO\_SMALL 1008  
ERR\_PILOT\_COPY\_FAILED 1008  
ERR\_PILOT\_FILE\_ALREADY\_EXISTS 1007  
ERR\_PILOT\_INVALID\_BUFFER 1006  
ERR\_PILOT\_INVALID\_CREATORID 1006  
ERR\_PILOT\_INVALID\_FILE\_INDEX 1007  
ERR\_PILOT\_INVALID\_FILE\_TYPE 1006  
ERR\_PILOT\_INVALID\_FILENAME 1007  
ERR\_PILOT\_INVALID\_INDEX 1006  
ERR\_PILOT\_INVALID\_PATH 1007  
ERR\_PILOT\_INVALID\_PATH\_TYPE 1007  
ERR\_PILOT\_INVALID\_REGISTRY 1007  
ERR\_PILOT\_INVALID\_SOURCE\_FILE 1006  
ERR\_PILOT\_INVALID\_USER 1008  
ERR\_PILOT\_INVALID\_USER\_INDEX 1008  
ERR\_PILOT\_NO\_DIRECTORY 1005  
ERR\_PILOT\_NO\_USER\_FILE 1005  
ERR\_PILOT\_NO\_USERS 1008  
ERR\_PILOT\_NO\_USERSDAT\_FILE 1008  
ERR\_PILOT\_NOT\_FOUND 1004  
ERR\_PILOT\_USER\_ACCESS 1004

ERR\_PILOT\_USERSDAT\_ALREADY\_EXISTS 1007  
ERR\_REGISTRY\_ACCESS 845  
ERR\_STORAGE\_ACCESS 845  
ERR UM BAD\_FILENAME 1090  
ERR UM BASE 1089  
ERR UM BUFSIZE\_TOO\_SMALL 1091  
ERR UM CANNOT\_WRITE\_TO\_STORE 1089  
ERR UM DEV\_CFG\_DATA\_NOT\_AVAILABLE 1088  
ERR UM FUNCTION\_NOT\_SUPPORTED 1088  
ERR UM INVALID\_BUFFER 1090  
ERR UM INVALID\_INDEX 1090  
ERR UM INVALID\_POINTER 1089  
ERR UM INVALID\_REGISTRY 1090  
ERR UM INVALID\_USER 1091  
ERR UM INVALID\_USER\_DIR 1090  
ERR UM INVALID\_USER\_INDEX 1091  
ERR UM INVALID\_USER\_NAME 1090  
ERR UM MULTIPLE\_USERS\_EXISTS 1088  
ERR UM NO\_CORE\_PATH 1089  
ERR UM NO\_DEFAULT\_USER 1088  
ERR UM NO\_DIRECTORY 1090  
ERR UM NO\_USER\_FOUND 1090  
ERR UM NO\_USERS 1091  
ERR UM NO\_USERSDAT\_FILE 1091  
ERR UM NOT\_FOUND 1090  
ERR UM OTHER\_USERSDAT\_ACCESS\_PROBLEM 1089  
ERR UM SAVE\_ERR 1090  
ERR UM SEMAPHORE\_ACCESS 1089  
ERR UM STRING\_TOO\_BIG 1088  
ERR UM SYNC\_PATH\_TOO\_BIG 1089  
ERR UM UNABLE\_TO\_CREATE\_NEW\_FILE 1089  
ERR UM USER\_ACCESS 1090  
ERR UM USER\_ALREADY\_EXISTS 1089  
ERR UM USER\_DIR\_ALREADY\_IN\_USE 1089  
ERR UM USERSDAT\_ALREADY\_EXISTS 1091  
ERR\_UNABLE\_TO\_CREATE\_CONDUIT 845  
ERR\_UNABLE\_TO\_DELETE 846  
ERR\_UNABLE\_TO\_INSTALL\_OLD 845  
ERR\_UNABLE\_TO\_SET\_CONDUIT\_VALUE 845  
ERR\_USER\_MANAGER\_BASE 1005

---

ERR\_VALUE\_NOT\_FOUND 845  
ERROR\_HSAPI\_ERROR\_BASE 949  
ERROR\_HSAPI\_FAILURE 949  
ERROR\_HSAPI\_HOTSYNC\_NOT\_FOUND 949  
ERROR\_HSAPI\_INVALID\_CONN\_TYPE 949  
ERROR\_HSAPI\_INVALID\_POINTER 950  
ERROR\_HSAPI\_INVALID\_STATUS\_FLAG 950  
ERROR\_HSAPI\_NO\_HOTSYNC\_PATH 950  
ERROR\_HSAPI\_REG\_FAILURE 949  
ERROR\_HSAPI\_UNABLE\_TO\_CLOSE 949  
ERROR\_HSAPI\_UNABLE\_TO\_START 950  
ERROR\_HSAPI\_UNKNOWN\_STATUS\_TYPE 949  
eRunAlways 505  
eSchema 417  
eSecure 417  
eSlow 426  
eStream 416  
eSyncPref 424  
eSyncPrefDoNotUse 424  
eSyncRecAttrDoNotUse 425  
eSyncRecAttrs 425  
eSyncTypeDoNotUse 427  
eSyncTypes 426  
eTemporaryPreference 424  
Expansion Manager  
    C API 547  
ExpansionMgr.h 547  
EXPAPI\_ERR\_CLASS 486  
expCapabilityHasStorage 551  
expCapabilityReadOnly 551  
ExpCardInfo() 555  
expCardInfoStringMaxLen 553  
ExpCardInfoType 549  
ExpCardPresent() 556  
expErrCardNoSectorReadWrite 560  
expErrCardNotPresent 560  
expErrEnumerationEmpty 560  
expErrInvalidSlotRefNumber 560  
expErrNotEnoughPower 560  
expErrNotOpen 560  
expErrSlotDeallocated 560  
expErrUnsupportedOperation 560  
expFtrIDVersion 553

expIteratorStart 550  
expIteratorStop 550  
ExpMediaType\_Any 552  
ExpMediaType\_CompactFlash 552  
ExpMediaType\_MemoryStick 552  
ExpMediaType\_MultiMediaCard 552  
ExpMediaType\_PlugNPlay 552  
ExpMediaType\_PoserHost 552  
ExpMediaType\_RAMDisk 552  
ExpMediaType\_SecureDigital 552  
ExpMediaType\_SmartMedia 552  
ExpSlotEnumerate() 557  
ExpSlotMediaType() 559  
extended databases  
    Sync Manager  
        C API 211

## F

FileInfoType 563  
FileInstallType 848, 953  
FILEINSTALLTYPE\_SIZE 664, 957  
FileOrigin 564  
FileRef 564  
FmDisableCurrentUserConduitByIndex() 819  
FmDisableCurrentUserConduitByPath() 821  
FmDisableSystemConduitByIndex() 823  
FmDisableSystemConduitByPath() 825  
FmEnableCurrentUserConduitByPath() 827  
FmEnableSystemConduitByPath() 829  
FmGetCurrentUserConduitByIndex() 831  
FmGetCurrentUserConduitCount() 832  
FmGetCurrentUserConduitFolder() 833  
FmGetCurrentUserDisabledConduitFolder() 835  
FmGetSystemConduitByIndex() 837  
FmGetSystemConduitCount() 838  
FmGetSystemConduitFolder() 839  
FmGetSystemDisabledConduitFolder() 841  
folder-based conduit registration  
    C API 651  
fsFilesystemType\_AFS 571  
fsFilesystemType\_EXT2 571  
fsFilesystemType\_FAT 571  
fsFilesystemType\_FFS 571

---

fsFilesystemType\_HFS 571  
fsFilesystemType\_HFSPlus 571  
fsFilesystemType\_HPFS 571  
fsFilesystemType\_MFS 571  
fsFilesystemType\_NFS 571  
fsFilesystemType\_Novell 571  
fsFilesystemType\_NTFS 571  
fsFilesystemType\_VFAT 571  
fsOriginBeginning 576  
fsOriginCurrent 576  
fsOriginEnd 576

## G

GetConduitInfo() 515  
GetConduitName() 518  
GetConduitVersion() 520  
GetNotifierVersion() 1100

## H

header file organization, Sync Manager 3  
HHToPC 1012  
HotSync log  
    C API 529  
HotSync Manager  
    C API 927  
HOTSYNC\_STATUS\_IDLE 934  
HOTSYNC\_STATUS\_SYNCING 934  
HS\_API\_VERSION\_1 929  
HS\_API\_VERSION\_2 929  
HS\_Notify() 1101  
HS\_SYNC\_FAILURE 1098  
HS\_SYNC\_QUERYOK 1098  
HS\_SYNC\_STARTED 1098  
HS\_SYNC\_SUCCESS 1098  
HS\_USER\_UPDATE 1098  
HSAPI.dll 927  
HSAPI.h 927  
HSAPP\_ERR\_CLASS 491  
HsCheckApiStatus() 936  
HsCloseApp 931  
HS\_CONNECTION\_DISABLED 929  
HS\_CONNECTION\_ENABLED 929  
HSConnectionType 930

HsDisplayCustomDlg() 937  
HsDisplayFileLink() 938  
HsDisplayLog() 939  
HsDisplaySetupDlg() 940  
HSFLAG\_DEVICE\_SYNC\_CHECK 932  
HSFLAG\_INSPECT\_CONDUIT 932  
HSFLAG\_LOG\_DEBUG\_LEVEL\_1 932  
HSFLAG\_LOG\_DEBUG\_LEVEL\_2 932  
HSFLAG\_NONE 932  
HSFLAG\_RESTORE\_REGISTRY 932  
HSFLAG\_RESTORE\_REGISTRY\_DEFAULT 932  
HSFLAG\_VERBOSE 933  
HsGetApiVersion() 941  
HsGetCommStatus() 942  
HsGetSyncStatus() 943  
HSLog.h 529  
HSLog20.dll 529  
HSNotify.h 1097  
HsRefreshConduitInfo() 944  
HsReserved 931  
HsResetComm() 945  
HsRestart 931  
HsSetAppStatus() 946  
HsSetCommStatus() 948  
HsStartApp 931  
HsStatusType 931

## I

ImGetDirectory() 853  
ImGetDWord() 855  
ImGetExtension() 857  
ImGetMask() 859  
ImGetModule() 860  
ImGetName() 862  
ImGetPrerequisites() 1127  
ImGetString() 864  
ImGetSystemDirectory() 866  
ImGetSystemDWord() 868  
ImGetSystemExtension() 870  
ImGetSystemMask() 872  
ImGetSystemModule() 873  
ImGetSystemName() 875  
ImGetSystemString() 877

---

ImRegister() 879  
ImRegisterID() 880  
ImRegisterSystem() 881  
ImRegisterSystemID() 883  
ImSetDirectory() 884  
ImSetDWord() 886  
ImSetExtension() 888  
ImSetMask() 890  
ImSetModule() 891  
ImSetName() 892  
ImSetPrerequisites() 1127  
ImSetString() 893  
ImSetSystemDirectory() 895  
ImSetSystemDWord() 897  
ImSetSystemExtension() 899  
ImSetSystemMask() 901  
ImSetSystemModule() 902  
ImSetSystemName() 903  
ImSetSystemString() 904  
ImUnregisterID() 906  
ImUnregisterSystemID() 907  
InstAide.dll 951  
InstAide.h 951  
INSTAIDE\_LIB\_VER\_MAJOR 957  
INSTAIDE\_LIB\_VER\_MINOR 957  
Install Conduit Manager  
    C API 847

## K

kOffsetEndOfData 431

## L

LANG\_DLL.h 539  
language resource DLL, loading 539  
LANGUAGE\_DUTCH 540  
LANGUAGE\_ENGLISH 540  
LANGUAGE\_FRENCH 540  
LANGUAGE\_GERMAN 540  
LANGUAGE\_ITALIAN 540  
LANGUAGE\_JAPANESE 540  
LANGUAGE\_PORTUGUESE 540

LANGUAGE\_SCHINESE 540  
LANGUAGE\_SPANISH 540  
LANGUAGE\_TCHINESE 540  
localization 539  
LogAddEntry() 535  
LogAddFormattedEntry() 536  
LogBuildRemoteLog() 1127  
LogCloseLog() 1127  
.LogError 538  
LogGetWorkFileName() 1127  
LogInit() 1127  
LogSaveLog() 1127  
LogTestCounters() 537  
LogUnInit() 1127

## M

MAX\_COM\_PORT 662  
MFC\_NOT\_USED 507  
MFC\_VERSION\_41 507  
MFC\_VERSION\_50 507  
MFC\_VERSION\_60 507  
MODEM\_COM\_PORT 662

## N

NmFind() 911  
NmFindSystem() 912  
NmGetByIndex() 913  
NmGetCount() 915  
NmGetSystemByIndex() 916  
NmGetSystemCount() 918  
NmRegister() 919  
NmRegisterSystem() 920  
NmRenameByIndex() 921  
NmRenameSystemByIndex() 923  
NmUnregister() 925  
NmUnregisterSystem() 926  
Notifier Install Manager  
    API 909

## O

OpenConduit() 521

---

**P**

Palm OS Common Language API

about 539

Palm OS version

retrieving

during a HotSync operation 396

outside of a HotSync operation 1035

PALM\_CMN.h 539

PALM\_DEF\_PRIORITY 506

PALM\_MAX\_PRIORITY 506

PALM\_MIN\_PRIORITY 506

PalmAboutBox() 1126

PalmCmn.dll 539

PalmFreeLanguage() 542

PalmGetResourceVersion() 543, 1126

PalmGetVersion() 544

PalmLoadLanguage() 545

Password Library 1093

password.h 1093

PASSWORD\_LENGTH 414

PCFGCONDUIT 511

PCONFIGURECONDUIT 513

PCToHH 1012

PGETCONDUITINFO 515

PGETCONDUITNAME 518

PGETCONDUITVERSION 520

PGETNOTIFIERVERSIONFUNC 1100

PHSNOTIFYFUNC 1101

Pi\*() functions 1127

PILOT\_PATH\_HOME 956

PILOT\_PATH\_HOTSYNC 956

PILOT\_PATH\_MAX 956

PILOT\_PATH\_TUTORIAL 956

PlmGetLibVersion() 961

PlmGetUserIDFromName() 962

PlmGetUserNameFromID() 963

PlmMoveInstallFileToHandheld() 964

PlmMoveInstallFileToSlot() 966

PlmSlotGetFileCount() 968

PlmSlotGetFileInfo() 969

PlmSlotInstallFile() 971

PlmSlotMoveInstallFile() 973

PlmSlotRemoveInstallFile() 975

PltGetFileCount() 976

PltGetFileInfo() 977

PltGetFileName() 979

PltGetFileTypeExtension() 981

PltGetFileTypesCount() 983

PltGetInstallConduitCount() 984

PltGetInstallConduitInfo() 985

PltGetInstallCreatorInfo() 986

PltGetInstallFileFilter() 987

PltGetInstallFileFilterForUser() 988

PltGetPath() 990

PltGetRegistryPath() 991

Plt GetUser() 992

Plt GetUserCount() 994

Plt GetUserDirectory() 995

PltInstallFile() 996

PltIsInstallMaskSet() 997

PltIsUserProfile() 998

PltRemoveInstallFile() 999

PltRemoveInstallRegistry() 1000

PltResetInstallMask() 1001

PltSetInstallRegistry() 1002

PltSetPath() 1003

plug and play media type 552

POOPENCONDUIT 521

private and obsolete functions, C API 1125

PRIVATE\_BIT 310

PROGRESSFN() 523

PsDelete() 1126

PsGetInteger() 1126

PsGetNameByIndex() 1126

PsGetNameCount() 1126

PsGetSectionByIndex() 1126

PsGetSectionCount() 1126

PsGetString() 1126

PsGetVersion() 1126

PsSetInteger() 1126

PsSetString() 1126

PWD5.lib 1093

PwdVerify() 1094

---

## R

registering notifiers  
    C API 909  
RegistrationInfoType 500  
REMOTE\_CARDNAMELEN 414  
REMOTE\_MANUENAMELEN 414  
REMOTE\_USERNAME 414  
revision history  
    C/C++ Sync Suite APIs 1105  
ROM  
    version. See *Palm OS version*  
RowID 27

## S

schema databases  
    Sync Manager  
        C API 5  
security  
    secure databases  
        creating 445  
SIZEOF\_DB\_CATEGORY\_DEFN 44  
SIZEOF\_DB\_COLUMN\_DEFN 44  
SIZEOF\_DB\_COLUMN\_REMOVE 44  
SIZEOF\_DB\_COLUMN\_VALUE 44  
SIZEOF\_DB\_DATABASE\_INFO 431  
SIZEOF\_DB\_ROW\_DATA 44  
SIZEOF\_DB\_ROW\_RESULT 44  
SIZEOF\_DB\_SYNC\_MODE 44  
SIZEOF\_DB\_TABLE\_DEFN 44  
slArchiveFailed 531  
slBadStream 538  
slCategoryDeleted 530  
slChangeCatFailed 530  
slCustomLabel 530  
slDateChanged 530  
slDeleteFileFailed 538  
slDoubleModify 530  
slDoubleModifyArchive 530  
slDoubleModifySubsc 531  
slError 532  
slFileLinkCompleted 531  
slFileLinkDeleted 531  
slHTMLText 532

slLocalAddFailed 531  
slLocalSaveFailed 531  
slMoveFileFailed 538  
slNoError 538  
slRecCountMismatch 531  
slRecommendation 532  
slRemoteAddFailed 530  
slRemoteChangeFailed 530  
slRemoteDeleteFailed 531  
slRemotePurgeFailed 530  
slRemoteReadFailed 530  
slResetFlagsFailed 531  
slReverseDelete 530  
slSyncAborted 531  
slSyncDidNothing 532  
slSyncFinished 531  
slSyncSessionCancelled 532  
slSyncSessionEnd 532  
slSyncSessionStart 532  
slSyncStarted 531  
slText 530  
slTooManyCategories 530  
slWarning 531  
slXMapFailed 531  
Sync Manager  
    C API 1  
        any database type 383  
        classic databases 289  
        extended databases 211  
        header file organization 3  
        schema databases 5  
        versions 3  
SYNC\_CLOSE\_DB\_OPT\_SYNCED\_ALL\_CHANGES 33  
SYNC\_CLOSE\_DB\_OPT\_UPDATE\_BACKUP\_DATE 410  
SYNC\_CLOSE\_DB\_OPT\_UPDATE\_MOD\_DATE 410  
SYNC\_DB\_INFO\_OPT\_GET\_ATTRIBUTES 412  
SYNC\_DB\_INFO\_OPT\_GET\_MAX\_REC\_SIZE 412  
SYNC\_DB\_INFO\_OPT\_GET\_SCHEMADB\_FIELDS 34  
SYNC\_DB\_INFO\_OPT\_GET\_SIZE 412

---

SYNC\_DB\_NAMELEN 429  
SYNC\_DB\_SRCH\_OPT\_NEW\_SEARCH 413  
SYNC\_DB\_SRCH\_OPT\_ONLY\_LATEST 413  
SYNC\_ERR\_CLASS 486  
SYNC\_FATAL\_ERR 492  
SYNC\_FATAL\_ERR\_MASK 491  
SYNC\_MAX\_HH\_LOG\_SIZE 429  
SYNC\_MAX\_PROD\_ID\_SIZE 429  
SYNC\_MAX\_USERNAME\_LENGTH 429  
SYNC\_REMOTE\_CARDNAME\_BUF\_SIZE 429  
SYNC\_REMOTE\_MANUENAME\_BUF\_SIZE 429  
SYNC\_REMOTE\_PASSWORD\_BUF\_SIZE 429  
SYNC\_REMOTE\_USERNAME\_BUF\_SIZE 429  
Sync20.dll 3, 5, 211, 289, 383  
SyncAddLogEntry() 437  
SYNCAPI\_VER\_MAJOR\_2 433  
SYNCAPI\_VER\_MINOR\_0 433  
SYNCAPI\_VER\_MINOR\_1 433  
SYNCAPI\_VER\_MINOR\_2 433  
SYNCAPI\_VER\_MINOR\_3 433  
SYNCAPI\_VER\_MINOR\_4 433  
SYNCAPI\_VER\_MINOR\_5 433  
SyncBackupDatabase() 438  
SyncBackupSecurityData() 442  
SyncCallApplication() 1128  
SyncCallDeviceApplication() 444  
SyncCallRemoteModule() 315  
SyncChangeCategory() 319  
SyncCloseDB() 321  
SyncCloseDBEx() 322  
SyncCommon.h 3, 383  
SyncCreateDB() 324  
SyncDatabaseInfoType 305  
SyncDb.h 3, 5  
SyncDbAddCategory() 56  
SyncDbAddColumns() 58  
SyncDbAddRowCategory() 61  
SyncDbAddTable() 63  
SyncDbCloseDatabase() 65  
SyncDbCreateDatabase() 67  
SyncDbCreateRow() 70  
SyncDbCreateRows() 73  
SyncDbDeleteAllRowsInTable() 76  
SyncDbDeleteDatabase() 78  
SyncDbDeleteRow() 79  
SyncDbDeleteRows() 81  
SyncDbDeleteRowsInCategory() 83  
SyncDbFindDatabase() 85  
SyncDbFindDatabaseByTypeCreator() 87  
SyncDbGetAllColumnValues() 89  
SyncDbGetCategoryCount() 92  
SyncDbGetCategoryDefinitionList() 93  
SyncDbGetChangeContext() 96  
SyncDbGetColumnDefinitions() 99  
SyncDbGetColumnIDs() 101  
SyncDbGetColumnPropertyValue() 104  
SyncDbGetColumnValue() 107  
SyncDbGetColumnValues() 110  
SyncDbGetModifiedCategoryDefinitionList() 113  
SyncDbGetModifiedTableInfoList() 116  
SyncDbGetRowCategory() 119  
SyncDbGetRowCountInTable() 122  
SyncDbGetSyncMode() 124  
SyncDbGetTableCount() 126  
SyncDbGetTableInfo() 127  
SyncDbGetTableInfoList() 129  
SyncDbGetTableSchema() 132  
SyncDbIsRowInCategory() 135  
SyncDbModifyCategory() 138  
SyncDbMoveRowsInCategory() 140  
SyncDbOpenDatabase() 143  
SyncDbPurgeAllRowsInTable() 145  
SyncDbReadModifiedRowInfoList() 147  
SyncDbReadModifiedRows() 150  
SyncDbReadOpenDatabaseInfo() 154  
SyncDbReadRow() 156  
SyncDbReadRowInfo() 158  
SyncDbReadRowInfoList() 160  
SyncDbReadRows() 163  
SyncDbReadRowsByRowInfo() 167  
SyncDbReleaseStorage() 170  
SyncDbRemoveCategory() 172  
SyncDbRemoveCategoryFromAllRows() 174  
SyncDbRemoveColumnProperty() 177  
SyncDbRemoveColumns() 179  
SyncDbRemoveRow() 181

---

SyncDbRemoveRowCategory() 183  
SyncDbRemoveRows() 185  
SyncDbRemoveSecretRowsInTable() 187  
SyncDbRemoveTable() 189  
SyncDbSetColumnPropertyValue() 191  
SyncDbSetRowCategory() 193  
SyncDbWriteColumnValue() 195  
SyncDbWriteColumnValues() 197  
SyncDbWriteRow() 199  
SyncDbWriteRows() 202  
SyncDeInit() 1126  
SyncDeleteAllResourceRec() 326  
SyncDeleteDB() 327  
SyncDeleteRec() 328  
SyncDeleteResourceRec() 330  
SyncDm.h 3, 211  
SyncDmChangeCategory() 216  
SyncDmCloseDatabase() 218  
SyncDmCloseDatabaseEx() 220  
SyncDmCreateDatabase() 222  
SyncDmDeleteAllResourceRecords() 224  
SyncDmDeleteDatabase() 226  
SyncDmDeleteRecord() 228  
SyncDmDeleteResourceRecord() 230  
SyncDmFindDatabase() 232  
SyncDmFindDatabaseByTypeCreator() 234  
SyncDmGetDatabaseRecordCount() 236  
SyncDmMaxRemoteRecordSize() 238  
SyncDmOpenDatabase() 239  
SyncDmPurgeAllRecords() 241  
SyncDmPurgeAllRecordsInCategory() 243  
SyncDmPurgeDeletedRecords() 245  
SyncDmReadAppInfoBlock() 247  
SyncDmReadNextModifiedRecord() 249  
SyncDmReadNextModifiedRecordIn-  
Category() 253  
SyncDmReadNextRecordInCategory() 256  
SyncDmReadOpenDatabaseInfo() 259  
SyncDmReadPositionXMap() 261  
SyncDmReadRecordByID() 263  
SyncDmReadRecordByIndex() 267  
SyncDmReadResourceRecordByIndex() 270  
SyncDmReadSortInfoBlock() 273  
SyncDmResetRecordIndex() 275  
SyncDmResetSyncFlags() 277  
SyncDmWriteAppInfoBlock() 279  
SyncDmWriteRecord() 281  
SyncDmWriteResourceRecord() 284  
SyncDmWriteSortInfoBlock() 287  
SyncEndOfSync() 1126  
SYNCERR\_ACCESS\_DENIED 491  
SYNCERR\_ARG\_MISSING 489  
SYNCERR\_BACKUP\_BIT\_NOT\_SET 209  
SYNCERR\_BAD\_ARG 490  
SYNCERR\_BAD\_ARG\_WRAPPER 489  
SYNCERR\_BAD\_OPERATION 489  
SYNCERR\_BUILT\_IN\_PROPERTY 208  
SYNCERR\_CATEGORY\_NAME\_NOT\_-  
SPECIFIED 209  
SYNCERR\_COLUMNID\_ALREADY\_EXISTS 208  
SYNCERR\_COMM\_NOT\_INIT 488  
SYNCERR\_DEVICE\_NOT\_CONNECTED 491  
SYNCERR\_DISK\_FULL 491  
SYNCERR\_FILE\_ALREADY\_EXIST 488  
SYNCERR\_FILE\_ALREADY\_OPEN 488  
SYNCERR\_FILE\_NOT\_FOUND 487  
SYNCERR\_FILE\_NOT\_OPEN 487  
SYNCERR\_FILE\_OPEN 487  
SYNCERR\_INVALID\_CATEGORYID 207  
SYNCERR\_INVALID\_COLUMNID 207  
SYNCERR\_INVALID\_COLUMNNAME 209  
SYNCERR\_INVALID\_COLUMNSIZE 209  
SYNCERR\_INVALID\_COLUMNSPEC 208  
SYNCERR\_INVALID\_COLUMNTYPE 208  
SYNCERR\_INVALID\_FILE\_HANDLE 208  
SYNCERR\_INVALID\_ID 209  
SYNCERR\_INVALID\_IMAGE 491  
SYNCERR\_INVALID\_MATCH\_MODE 209  
SYNCERR\_INVALID\_PROPID 207  
SYNCERR\_INVALID\_TABLEDEFN 208  
SYNCERR\_INVALID\_TABLENAME 207  
SYNCERR\_LIMIT\_EXCEEDED 490  
SYNCERR\_LOCAL\_BUFF\_TOO\_SMALL 489  
SYNCERR\_LOCAL\_CANCEL\_SYNC 492  
SYNCERR\_LOCAL\_MEM 490  
SYNCERR\_LOST\_CONNECTION 492

---

SYNCERR\_MAX\_CATEGORY\_LIMIT 207  
SYNCERR\_MORE 487  
SYNCERR\_NAME\_ALREADY\_EXISTS 208  
SYNCERR\_NO\_DATA 208  
SYNCERR\_NO\_FILES\_OPEN 488  
SYNCERR\_NONE 486  
SYNCERR\_NOT\_FOUND 487  
SYNCERR\_NOT\_RECORDDB 207  
SYNCERR\_NOT\_SCHEMADB 207  
SYNCERR\_NOT\_SECUREDB 207  
SYNCERR\_ONE\_OR\_MORE\_FAILED 208  
SYNCERR\_OPERATION\_ABORTED 208  
SYNCERR\_PATH\_NOT\_FOUND 491  
SYNCERR\_READ\_ONLY 488  
SYNCERR\_RECORD\_BUSY 487  
SYNCERR\_RECORD\_DELETED 488  
SYNCERR\_REMOTE\_BAD\_ARG 489  
SYNCERR\_REMOTE\_CANCEL\_SYNC 492  
SYNCERR\_REMOTE\_MEM 489  
SYNCERR\_REMOTE\_NO\_SPACE 490  
SYNCERR\_REMOTE\_PASSWORD 209  
SYNCERR\_REMOTE\_SYS 490  
SYNCERR\_ROM\_BASED 488  
SYNCERR\_ROWS\_EXIST 208  
SYNCERR\_STREAM\_ERR 209  
SYNCERR\_TABLE\_NAME\_NOT\_SPECIFIED 208  
SYNCERR\_TOO\_MANY\_FILES 492  
SYNCERR\_TOO\_MANY\_OPEN\_FILES 492  
SYNCERR\_UNKNOWN 487  
SYNCERR\_UNKNOWN\_REQUEST 491  
SyncFindDbByName() 332  
SyncFindDbByNameParams 307  
SyncFindDbByTypeCreator() 334  
SyncFindDbByTypeCreatorParams 308  
SyncGenerateBackupFileName() 448  
SyncGetAPIVersion() 450  
SyncGetDBRecordCount() 336  
SyncGetDesktopTrustStatus() 451  
SyncGetHHOSVersion() 452  
SyncHHToHostDWord() 453  
SyncHHToHostWord() 454  
SyncHostToHHDWord() 455  
SyncHostToHHWord() 456

Synchronize 1012  
SyncInit() 1126  
SyncInstallAndBackupDatabase() 457  
SyncInstallDatabase() 460  
SyncIsDatabaseBackupNeeded() 462  
SyncLoopBackTest() 466, 1126  
SyncMaxRemoteRecSize() 337  
SyncMgr.h 3, 289  
SyncOpenDB() 339  
SyncPreSendCmd() 1126  
SyncPurgeAllRecs() 341  
SyncPurgeAllRecsInCategory() 342  
SyncPurgeDeletedRecs() 344  
SyncReadAppPreference() 345  
SyncReadBackupImageInfo() 467  
SyncReadDatabaseList() 205  
SyncReadDBAppInfoBlock() 347  
SyncReadDBList() 349  
SyncReadDBSortInfoBlock() 351  
SyncReadFeature() 469  
SyncReadNetSyncInfo() 1126  
SyncReadNextModifiedRec() 353  
SyncReadNextModifiedRecInCategory() 355  
SyncReadNextRecInCategory() 357  
SyncReadOpenDbInfo() 359  
SyncReadOpenDbInfoParams 309  
SyncReadPositionXMap() 361  
SyncReadProdCompInfo() 1126  
SyncReadRecordById() 364  
SyncReadRecordByIndex() 366  
SyncReadResRecordByIndex() 368  
SyncReadSingleCardInfo() 471  
SyncReadSysDateTime() 473  
SyncReadSystemInfo() 474  
SyncReadUserID() 475  
SyncRebootSystem() 476  
SyncRegisterConduit() 477  
SyncResetRecordIndex() 370  
SyncResetSyncFlags() 372  
SyncRestoreSecurityData() 478  
SYNCROMVMAJOR() 480  
SYNCROMVMINOR() 481  
SyncUnRegisterConduit() 482

---

SyncWriteAppPreference() 374  
SyncWriteDBAppInfoBlock() 376  
SyncWriteDBSortInfoBlock() 378  
SyncWriteNetSyncInfo() 1126  
SyncWriteRec() 380  
SyncWriteResourceRec() 382  
SyncWriteSysDateTime() 483  
SyncWriteUserID() 1126  
SyncYieldCycles() 485  
sysFileCExpansionMgr 553  
sysFileCVFSMgr 574  
sysFileTSimulator 579  
sysFileTSlotDriver 579  
SZ\_CFGCONDUITINFO 508  
SZ\_CONDUITREQUESTINFO 508  
SZ\_REGISTRATIONINFO 508

## T

TRANS\_ERR\_CLASS 486

## U

Ui\*() functions 1127  
UM\_LIB\_VER\_MAJOR 1014  
UM\_LIB\_VER\_MINOR 1014  
UmAddUser() 1019  
UmClearInstallMask() 1021  
UmDeleteKey() 1023  
UmDeleteUser() 1025  
UmDeleteUserPermSyncPreferences() 1026  
UmDeleteUserTempSyncPreferences() 1028  
UmGetIDFromName() 1030  
UmGetIDFromPath() 1032  
UmGetInteger() 1034  
UmGetLibVersion() 1036  
UmGetRootDirectory() 1037  
UmGetString() 1039  
Um GetUserCount() 1041  
Um GetUserDirectory() 1042  
Um GetUserID() 1044  
Um GetUserName() 1045  
Um GetUserPassword() 1047  
Um GetUserPermSyncPreferences() 1049  
Um GetUserTempSyncPreferences() 1051

UmIsInstallMaskSet() 1053  
UmIsUserInstalled() 1055  
UmIsUserNameValid() 1057  
UmIsUserProfile() 1058  
UmRemoveUserTempSyncPreferences() 1059  
UmSetInstallMask() 1061  
UmSetInteger() 1063  
UmSetString() 1065  
UmSetUserDirectory() 1067  
UmSetUserInstall() 1069  
UmSetUserName() 1071  
UmSetUserPassword() 1128  
UmSetUserPermSyncPreferences() 1073  
UmSetUserTempSyncPreferences() 1075  
UmSlotGetDisplayName() 1077  
UmSlotGetExpMgrVersion() 1079  
UmSlotGetInfo() 1080  
UmSlotGetInstallDirectory() 1082  
UmSlotGetMediaType() 1084  
UmSlotGetSlotCount() 1086  
UmUserSyncAction 1012  
user data store  
    C API 1009  
UserData.dll 1009  
UserData.h 1009

## V

VFS Manager  
    C API 561  
VFSAnyMountParamPtr 565  
VFSAnyMountParamType 565  
VFSAPI.dll 547  
VFSAPI\_ERR\_CLASS 486  
VFSAPI\_VER\_MAJOR\_1 577  
VFSAPI\_VER\_MINOR\_0 577  
VFSAPI\_VER\_MINOR\_1 577  
VFSCustomControl() 584  
VFSDeinit() 1127  
VFSDirCreate() 586  
VFSDirEntryEnumerate() 588  
VFSErr.h 547, 561  
vfsErrBadData 646  
vfsErrBadName 646

---

vfsErrBufferOverflow 645  
vfsErrDirectoryNotFound 647  
vfsErrDirNotEmpty 646  
vfsErrDiskFileAccess 647  
vfsErrDiskFull 647  
vfsErr FileAccessOther 647  
vfsErrFileAlreadyExists 645  
vfsErrFileBadRef 645  
vfsErrFileEOF 646  
vfsErrFileGeneric 645  
vfsErrFileNotFoundException 646  
vfsErrFilePermissionDenied 645  
vfsErrFileStillOpen 645  
vfsErrInvalidOperation 647  
vfsErrIsADirectory 647  
vfsErrNameShortened 647  
vfsErrNoFileSystem 646  
vfsErrNotADirectory 646  
vfsErrUnimplemented 646  
vfsErrVolumeBadRef 646  
vfsErrVolumeFull 646  
vfsErrVolumeStillMounted 646  
VFSExportDatabaseToFile() 591  
VFSExportDatabaseToFileEx() 593  
vfsFileAttrArchive 572  
vfsFileAttrDirectory 572  
vfsFileAttrHidden 572  
vfsFileAttrLink 572  
vfsFileAttrReadOnly 572  
vfsFileAttrSystem 572  
vfsFileAttrVolumeLabel 572  
VFSFileClose() 596  
VFSFileCreate() 597  
vfsFileDateAccessed 570  
vfsFileDateCreated 570  
vfsFileDateModified 570  
VFSFileDelete() 599  
VFSFileEOF() 601  
VFSFileGet() 602  
VFSFileGetAttributes() 604  
VFSFileGetDate() 605  
VFSFileOpen() 607  
VFSFilePut() 609  
VFSFileRead() 611  
VFSFileRename() 613  
VFSFileResize() 615  
VFSFileSeek() 616  
VFSFileSetAttributes() 618  
VFSFileSetDate() 619  
VFSFileSize() 621  
VFSFileTell() 622  
VFSFileWrite() 623  
vfsFtrIDVersion 574  
VFSGetAPIVersion() 625  
VFSGetDefaultDirectory() 626  
VFSImportDatabaseFromFile() 629  
VFSImportDatabaseFromFileEx() 631  
VFSInit() 1127  
vfsInvalidFileRef 573  
vfsInvalidSlotLibRefNum 573  
vfsInvalidVolRef 573  
VFSMgr.h 561  
vfsModeCreate 575  
vfsModeExclusive 575  
vfsModeRead 575  
vfsModeReadWrite 575  
vfsModeTruncate 575  
vfsModeWrite 575  
vfsMountClass\_Simulator 579  
vfsMountClass\_SlotDriver 579  
vfsMountFlagsReserved1 580  
vfsMountFlagsReserved2 580  
vfsMountFlagsReserved3 580  
vfsMountFlagsReserved4 580  
vfsMountFlagsReserved5 580  
vfsMountFlagsUseThisFileSystem 580  
VFSSlotMountParamType 566  
VFSSupport() 633  
vfsVolumeAttrHidden 578  
vfsVolumeAttrReadOnly 578  
vfsVolumeAttrSlotBased 578  
VFSVolumeEnumerate() 634  
VFSVolumeFormat() 636  
VFSVolumeGetLabel() 639  
VFSVolumeInfo() 641  
VFSVolumeSetLabel() 642  
VFSVolumeSize() 644  
VolumeInfoType 567