

# Palm OS® Programmer's API Reference

Palm OS® 5 SDK

## CONTRIBUTORS

Written by Greg Wilson, Jean Ostrem, Clif Liu, and Doug Fulton

Engineering contributions by Jesse Donaldson, Noah Gibbs, Lee Taylor, Danny Epstein, Peter Epstein, Ludovic Ferrandis, Gilles Fabre, David Fedor, Roger Flores, Steve Lemke, Bob Ebert, Ken Krugler, Paul Plaquette, Bruce Thompson, Tim Wiegman, Gavin Peacock, Ryan Robertson, and Waddah Kudaimi

Copyright © 1996 - 2002, PalmSource, Inc. and its affiliates. All rights reserved. This documentation may be printed and copied solely for use in developing products for Palm OS® software. In addition, two (2) copies of this documentation may be made for archival and backup purposes. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form or by any means or used to make any derivative work (such as translation, transformation or adaptation) without express written consent from PalmSource, Inc.

PalmSource, Inc. reserves the right to revise this documentation and to make changes in content from time to time without obligation on the part of PalmSource, Inc. to provide notification of such revision or changes.

PALMSOURCE, INC. AND ITS SUPPLIERS MAKE NO REPRESENTATIONS OR WARRANTIES THAT THE DOCUMENTATION IS FREE OF ERRORS OR THAT THE DOCUMENTATION IS SUITABLE FOR YOUR USE. THE DOCUMENTATION IS PROVIDED ON AN "AS IS" BASIS. PALMSOURCE, INC. AND ITS SUPPLIERS MAKE NO WARRANTIES, TERMS OR CONDITIONS, EXPRESS OR IMPLIED, EITHER IN FACT OR BY OPERATION OF LAW, STATUTORY OR OTHERWISE, INCLUDING WARRANTIES, TERMS, OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND SATISFACTORY QUALITY. TO THE FULL EXTENT ALLOWED BY 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, OR PUNITIVE DAMAGES OF ANY KIND, OR FOR LOSS OF REVENUE OR PROFITS, LOSS OF BUSINESS, LOSS OF INFORMATION OR DATA, OR OTHER FINANCIAL LOSS ARISING OUT OF OR IN CONNECTION WITH THIS DOCUMENTATION, EVEN IF PALMSOURCE, INC. OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Palm OS, Palm Computing, HandFAX, HandSTAMP, HandWEB, Graffiti, HotSync, iMessenger, MultiMail, Palm.Net, PalmPak, PalmConnect, PalmGlove, PalmModem, PalmPoint, PalmPrint, and PalmSource are registered trademarks of PalmSource, Inc. or its affiliates. Palm, the Palm logo, MyPalm, PalmGear, PalmPix, PalmPower, AnyDay, EventClub, HandMAIL, the HotSync logo, PalmGlove, Palm Powered, the Palm trade dress, Smartcode, Simply Palm, ThinAir, WeSync, and Wireless Refresh 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 DOCUMENTATION IS PROVIDED ON A COMPACT DISC, THE OTHER SOFTWARE AND DOCUMENTATION ON THE COMPACT DISC ARE SUBJECT TO THE LICENSE AGREEMENT ACCOMPANYING THE COMPACT DISC.

Palm OS Programmer's API Reference  
Document Number 3003-005  
May 13, 2002  
For the latest version of this document, visit  
<http://www.palmos.com/dev/support/docs/>.

PalmSource, Inc.  
5470 Great America Pkwy.  
Santa Clara, CA 95054  
USA  
[www.palmos.com](http://www.palmos.com)

# Table of Contents

---

<b>About This Document</b>	<b>xvii</b>
Palm OS SDK Documentation . . . . .	xvii
What This Volume Contains . . . . .	xvii
Additional Resources . . . . .	xviii
Conventions Used in This Guide . . . . .	xix
 <b>Part I: User Interface</b>	
<b>1 Application Launch Codes</b>	<b>3</b>
Launch Codes . . . . .	6
Launch Flags. . . . .	36
<b>2 Palm OS Events</b>	<b>39</b>
Event Data Structures . . . . .	40
Event Reference . . . . .	43
<b>3 Notifications</b>	<b>71</b>
Notification Data Structures . . . . .	74
Notification Reference. . . . .	75
<b>4 Attention Manager</b>	<b>105</b>
Attention Manager Data Structures . . . . .	105
Attention Manager Constants . . . . .	115
Attention Manager Functions . . . . .	119
Application-Defined Functions . . . . .	130
<b>5 Categories</b>	<b>133</b>
Category Data Structures . . . . .	133
Category Constants . . . . .	134
Category Functions . . . . .	136
<b>6 Clipboard</b>	<b>153</b>
Clipboard Data Structures . . . . .	153
Clipboard Functions . . . . .	154

---

<b>7 Controls</b>	<b>157</b>
Control Data Structures . . . . .	157
Control Resources . . . . .	167
Control Functions. . . . .	167
<b>8 Date and Time Selector</b>	<b>185</b>
Date and Time Selections Data Structures . . . . .	185
Date and Time Selection Functions . . . . .	186
<b>9 Fields</b>	<b>195</b>
Field Data Structures . . . . .	195
Field Resources. . . . .	204
Field Functions. . . . .	204
<b>10 Find</b>	<b>249</b>
Find Functions . . . . .	249
<b>11 Forms</b>	<b>255</b>
Form Data Structures . . . . .	255
Form Constants . . . . .	274
Form Resources . . . . .	274
Form Functions. . . . .	275
Application-Defined Functions . . . . .	325
<b>12 Graffiti Shift</b>	<b>331</b>
GraffitiShift Functions. . . . .	331
<b>13 Insertion Point</b>	<b>335</b>
Insertion Point Functions . . . . .	335
<b>14 Lists</b>	<b>339</b>
List Data Structures . . . . .	339
List Resources . . . . .	343
List Functions . . . . .	343
Application-Defined Function . . . . .	355

---

<b>15 Menus</b>	<b>359</b>
Menu Data Structures . . . . .	359
Menu Constants . . . . .	370
Menu Resources . . . . .	371
Menu Functions . . . . .	371
<b>16 Private Records</b>	<b>389</b>
Private Record Data Structures . . . . .	389
Private Record Functions . . . . .	390
<b>17 Progress Manager</b>	<b>393</b>
Progress Manager Functions . . . . .	393
Application-Defined Functions . . . . .	400
<b>18 Scroll Bars</b>	<b>405</b>
Scroll Bar Data Structures . . . . .	405
Scroll Bar Resources. . . . .	410
Scroll Bar Functions. . . . .	410
<b>19 System Dialogs</b>	<b>417</b>
System Dialog Functions . . . . .	417
<b>20 Tables</b>	<b>419</b>
Table Data Structures . . . . .	419
Table Constants . . . . .	432
Table Resource . . . . .	432
Table Functions. . . . .	433
Application-Defined Functions . . . . .	475
<b>21 UI Color List</b>	<b>479</b>
UI Color Data Types . . . . .	479
UI Color Functions . . . . .	483
<b>22 UI Controls</b>	<b>489</b>
UI Control Functions . . . . .	489

---

<b>23 Miscellaneous User Interface Functions</b>	<b>493</b>
Miscellaneous User Interface Data Structures . . . . .	493
Miscellaneous User Interface Functions . . . . .	498
 <b>Part II: System Management</b>	
<b>24 Alarm Manager</b>	<b>505</b>
Alarm Manager Functions . . . . .	505
Application-Defined Functions . . . . .	509
<b>25 Bitmaps</b>	<b>513</b>
Bitmap Data Structures . . . . .	513
Bitmap Constants. . . . .	535
Bitmap Resources. . . . .	535
Bitmap Functions. . . . .	536
<b>26 Character Attributes</b>	<b>555</b>
Character Attribute Functions . . . . .	555
<b>27 Data and Resource Manager</b>	<b>561</b>
Data Manager Data Structures . . . . .	561
Data Manager Constants. . . . .	562
Data Manager Functions. . . . .	571
Application-Defined Functions . . . . .	640
<b>28 Error Manager</b>	<b>643</b>
ERROR_CHECK_LEVEL Define . . . . .	643
Error Manager Data Structures . . . . .	644
Error Manager Functions . . . . .	644
<b>29 Expansion Manager</b>	<b>653</b>
Expansion Manager Data Structures. . . . .	653
Expansion Manager Constants . . . . .	654
Expansion Manager Functions . . . . .	656
<b>30 Feature Manager</b>	<b>665</b>
Feature Manager Functions . . . . .	665

---

<b>31 File Streaming</b>	<b>673</b>
File Streaming Constants . . . . .	673
File Streaming Functions. . . . .	675
File Streaming Error Codes. . . . .	693
<b>32 Float Manager</b>	<b>695</b>
Float Manager Data Structures . . . . .	695
Float Manager Functions . . . . .	697
<b>33 Fonts</b>	<b>709</b>
Font Data Structures . . . . .	709
Font Constants . . . . .	717
Font Resources . . . . .	718
Font Functions . . . . .	723
<b>34 Graffiti Manager</b>	<b>737</b>
Graffiti Manager Functions . . . . .	737
<b>35 Helper API</b>	<b>749</b>
Helper Data Structures . . . . .	749
Helper Constants . . . . .	756
<b>36 Key Manager</b>	<b>759</b>
Key Manager Functions . . . . .	759
<b>37 Locale Manager</b>	<b>763</b>
Locale Manager Data Types . . . . .	763
Locale Manager Constants. . . . .	765
Locale Manager Functions . . . . .	768
<b>38 Memory Manager</b>	<b>775</b>
Memory Manager Functions . . . . .	775
<b>39 Notification Manager</b>	<b>801</b>
Notification Constants. . . . .	801
Notification Functions. . . . .	802
Application-Defined Functions . . . . .	810

---

<b>40 Overlay Manager</b>	<b>811</b>
Overlay Manager Data Structures . . . . .	811
Overlay Manager Constants . . . . .	812
Overlay Manager Functions . . . . .	813
<b>41 Password</b>	<b>823</b>
Password Functions . . . . .	823
<b>42 Pen Manager</b>	<b>825</b>
Pen Manager Functions . . . . .	825
<b>43 Preferences</b>	<b>827</b>
Preferences Data Types . . . . .	827
Preferences Constants . . . . .	839
Preferences Functions . . . . .	842
<b>44 Rectangles</b>	<b>853</b>
Rectangle Data Structures . . . . .	853
Rectangle Functions . . . . .	854
<b>45 Sound Manager</b>	<b>859</b>
Overview . . . . .	859
Simple Sound Structures and Constants . . . . .	860
Simple Sound Functions . . . . .	868
Simple Sound Application-Defined Functions . . . . .	877
Sampled Sound Structures, Constants, and Data Types . . . . .	879
Sampled Sound Functions . . . . .	884
Sampled Sound Application-Defined Functions . . . . .	896
<b>46 Standard IO</b>	<b>899</b>
Standard IO Functions . . . . .	899
Standard IO Provider Functions . . . . .	914
Application-Defined Function . . . . .	918
<b>47 String Manager</b>	<b>919</b>
String Manager Functions . . . . .	919

---

<b>48 System Event Manager</b>	<b>941</b>
System Event Manager Data Structures . . . . .	941
System Event Manager Functions . . . . .	942
<b>49 System Manager</b>	<b>961</b>
System Manager Data Structures . . . . .	961
System Functions . . . . .	962
Application-Defined Functions . . . . .	995
<b>50 Text Manager</b>	<b>997</b>
Text Manager Data Structures . . . . .	997
Text Manager Functions . . . . .	998
<b>51 Text Services Manager</b>	<b>1041</b>
Text Services Manager Data Structures. . . . .	1041
Text Services Manager Functions . . . . .	1042
<b>52 Time Manager</b>	<b>1045</b>
Time Manager Data Structures . . . . .	1045
Time Manager Constants . . . . .	1054
Time Manager Functions . . . . .	1055
<b>53 Virtual File System Manager</b>	<b>1075</b>
VFS Manager Data Structures . . . . .	1075
VFS Manager Constants . . . . .	1079
VFS Manager Functions . . . . .	1085
Application-Defined Functions . . . . .	1145
<b>54 Windows</b>	<b>1147</b>
Window Data Structures. . . . .	1147
Window Constants . . . . .	1163
Window Functions . . . . .	1164

---

<b>55 Miscellaneous System Functions</b>	<b>1247</b>
<b>Part III: Communications</b>	
<b>56 Connection Manager</b>	<b>1263</b>
Connection Manager Data Types . . . . .	1263
Connection Manager Constants. . . . .	1263
Connection Manager Functions. . . . .	1273
<b>57 Exchange Manager</b>	<b>1297</b>
Exchange Manager Data Structures . . . . .	1297
Exchange Manager Constants . . . . .	1306
Exchange Manager Functions . . . . .	1309
Application-Defined Functions . . . . .	1352
<b>58 Exchange Library</b>	<b>1357</b>
Exchange Library Functions . . . . .	1357
<b>59 IR Library</b>	<b>1373</b>
IR Library Data Structures . . . . .	1373
IR Library Constants . . . . .	1378
IR Stack Callback Events. . . . .	1380
IR Library Functions . . . . .	1383
IAS Functions . . . . .	1399
Application-Defined Functions . . . . .	1409
<b>60 Modem Manager</b>	<b>1411</b>
Modem Manager Functions . . . . .	1411
<b>61 Net Library</b>	<b>1413</b>
Net Library Data Structures . . . . .	1413
Net Library Constants. . . . .	1420
Net Library Functions . . . . .	1422
<b>62 Network Utilities</b>	<b>1507</b>
Network Utility Functions . . . . .	1507

---

<b>63 Script Plugin</b>	<b>1511</b>
Script Plugin Data Types . . . . .	1511
Script Plugin Constants . . . . .	1516
Script Plugin Functions . . . . .	1518
<b>64 Virtual Drivers</b>	<b>1523</b>
Driver Data Structures . . . . .	1523
Driver Constants . . . . .	1535
Virtual Driver-Defined Functions . . . . .	1538
Serial Manager Queue Functions . . . . .	1546
<b>65 Serial Manager</b>	<b>1551</b>
Serial Manager Data Structures . . . . .	1551
Serial Manager Constants . . . . .	1557
Serial Manager Functions . . . . .	1563
Serial Manager Application-Defined Functions . . . . .	1590
<b>66 Old Serial Manager</b>	<b>1593</b>
Serial Manager Data Structures . . . . .	1593
Serial Manager Functions . . . . .	1595
<b>67 Serial Link Manager</b>	<b>1611</b>
Serial Link Manager Functions . . . . .	1611
<b>68 Telephony Basic Services</b>	<b>1621</b>
Telephony Service Types . . . . .	1621
Telephony Data Structures . . . . .	1623
Telephony Constants . . . . .	1630
Telephony Functions . . . . .	1637
Feature Support Functions . . . . .	1676
<b>69 Telephony Security and Configuration</b>	<b>1681</b>
Telephony Security and Configuration Data Structures . . . . .	1681
Telephony Security and Configuration Constants . . . . .	1684
Telephony Security and Configuration Functions . . . . .	1684

---

<b>70 Telephony Network</b>	<b>1695</b>
Telephony Network Data Structures . . . . .	1695
Telephony Network Constants . . . . .	1698
Telephony Network Functions . . . . .	1699
<b>71 Telephony Calls</b>	<b>1715</b>
Telephony Calls Data Structures . . . . .	1715
Telephony Calls Functions . . . . .	1720
<b>72 Telephony SMS</b>	<b>1755</b>
Telephony SMS Data Structures . . . . .	1755
Telephony SMS Constants . . . . .	1781
Telephony SMS Functions . . . . .	1784
<b>73 Telephony Phone Book</b>	<b>1815</b>
Telephony Phone Book Data Structures . . . . .	1815
Telephony Phone Book Constants . . . . .	1820
Telephony Phone Book Functions . . . . .	1821
<b>Part IV: Libraries</b>	
<b>74 Internet Library</b>	<b>1839</b>
Internet Library Data Structures . . . . .	1839
Internet Library Constants . . . . .	1850
Internet Library Functions . . . . .	1853
<b>75 PalmOSGlue Library</b>	<b>1891</b>
PalmOSGlue Functions . . . . .	1892
<b>76 Bluetooth Library: General Functions</b>	<b>1925</b>
Security Functions . . . . .	1925
Utility Functions . . . . .	1929
<b>77 Bluetooth Library: Management</b>	<b>1939</b>
Bluetooth Management Data Structures . . . . .	1940
Management Callback Events . . . . .	1948
Management Event Status Codes . . . . .	1957

---

Library Management Functions . . . . .	1959
Management Functions . . . . .	1962
Application-Defined Functions . . . . .	1989
<b>78 Bluetooth Library: Sockets and Service Discovery</b>	<b>1991</b>
Socket-Related Data Structures . . . . .	1992
Socket Callback Events . . . . .	2011
Socket Disconnection Error Codes . . . . .	2023
Socket Functions . . . . .	2024
Service Discovery Protocol Functions . . . . .	2041
Application-Defined Functions . . . . .	2076
<b>79 Cryptography Provider Manager</b>	<b>2079</b>
The Default Provider . . . . .	2079
Fundamental CPM Functions . . . . .	2080
Using the Crypto-Info Structures . . . . .	2080
Using the Export Functions . . . . .	2081
CPM and AP Constants . . . . .	2082
CPM and AP Structures and Data Types . . . . .	2091
CPM Functions . . . . .	2098
CPM Error Codes . . . . .	2132
<b>80 SSL Functions</b>	<b>2135</b>
SSL Attribute Functions and Macros . . . . .	2136
A Note on the Function Names . . . . .	2136
SSL Library Functions . . . . .	2137
Application-Defined Functions . . . . .	2158
<b>81 SSL Structures and Data Types</b>	<b>2163</b>
SSL Data Types . . . . .	2163
SSL Structures . . . . .	2166
<b>82 SSL Attributes and Macros</b>	<b>2181</b>
SSL Macro Names . . . . .	2181
SSL Attribute Data Types . . . . .	2182
SSL Macro Pseudo-Protocol . . . . .	2183

---

SSL Attributes . . . . .	2187
SSL Attribute Constants . . . . .	2210
<b>83 SSL Error Codes</b>	<b>2213</b>
SSL Function Protocol Errors . . . . .	2213
SSL Alerts . . . . .	2214
SSL Handshake Errors. . . . .	2215
SSL Cryptography Errors . . . . .	2215
SSL Illegal Message Errors . . . . .	2216
SSL Certificate Errors . . . . .	2216
<b>84 SMS Exchange Library</b>	<b>2219</b>
SMS Exchange Library Data Structures . . . . .	2219
SMS Exchange Library Constants . . . . .	2232
<b>85 Personal Data Interchange Library</b>	<b>2237</b>
PDI Library Data Structures . . . . .	2237
PDI Library Constants. . . . .	2241
PDI Library Functions. . . . .	2253
<b>86 Unified Data Access Manager</b>	<b>2279</b>
UDA Manager Data Structures . . . . .	2279
UDA Manager Constants . . . . .	2283
UDA Manager Functions . . . . .	2284
UDA Object Creation Functions . . . . .	2291
<b>Part V: Appendixes</b>	
<b>A System Use Only Functions</b>	<b>2297</b>
<b>B Compatibility Guide</b>	<b>2303</b>
2.0 New Feature Set. . . . .	2304
3.0 New Feature Set. . . . .	2308
3.1 New Feature Set. . . . .	2312
3.2 New Feature Set. . . . .	2315
International Feature Set. . . . .	2316

---

Japanese Feature Set . . . . .	2318
Wireless Internet Feature Set . . . . .	2318
New Serial Manager Feature Set . . . . .	2320
Connection Manager Feature Set . . . . .	2323
3.5 New Feature Set . . . . .	2324
Notification Feature Set . . . . .	2330
4.0 New Feature Set . . . . .	2330
Expansion Manager Feature Set . . . . .	2336
VFS Manager Feature Set . . . . .	2337
Bluetooth Library Feature Set . . . . .	2339
High-Density Display Feature Set . . . . .	2342
Sound Stream Feature Set . . . . .	2344
5.0 New Feature Set . . . . .	2346
5.1 New Feature Set . . . . .	2351
<b>C 1.0 Float Manager</b>	<b>2355</b>
Float Manager Functions . . . . .	2355
<b>Index</b>	<b>2363</b>



# About This Document

---

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

## Palm OS SDK Documentation

The following documents are part of the SDK:

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

## What This Volume Contains

This section provides an overview of this volume.

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

## About This Document

### *Additional Resources*

---

- Part II, “System Management,” documents the API contained in the header files in the \Inc\Core\System\ folder. This part contains chapters covering subjects such as the alarm manager, data and resource manager, feature manager, float manager, graffiti manager, key manager, memory manager, preferences manager, sound manager, string manager, and system manager.
- Part III, “Communications,” documents the API related to communications, such as the exchange manager, IR library, net library, Secure Sockets Layer (SSL) library, serial manager, and serial drivers.
- Part IV, “Libraries,” documents the API contained in the header files in the \Inc\Libraries\ folder. This part contains chapters covering the Internet Library and the Palm OS Glue library.

## Additional Resources

- Documentation

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

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

- Training

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

<http://www.palmsos.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.palmsos.com/dev/support/kb/>

## Conventions Used in This Guide

This guide uses the following typographical conventions:

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

## **About This Document**

*Conventions Used in This Guide*

---

# **Part I: User Interface**



# Application Launch Codes

---

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

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

- [Launch Codes](#)
- [Launch Flags](#)

To learn what a launch code is and how to use it, see the chapter titled “[Application Startup and Stop](#)” in the *Palm OS Programmer’s Companion*, vol. I.

**Table 1.1 Palm OS Launch Codes**

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

## Application Launch Codes

---

**Table 1.1 Palm OS Launch Codes (*continued*)**

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

**Table 1.1 Palm OS Launch Codes (*continued*)**

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

## Application Launch Codes

### Launch Codes

---

## Launch Codes

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

### **sysAppLaunchCmdAddRecord**

Add a record to an application's database.

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

For more information on sending messages via iMessenger, see “[Sending Email Messages](#)” on page 210 in the *Palm OS Programmer's Companion*, vol. II, *Communications*.

---

**IMPORTANT:** Implemented for iMessenger only if [Wireless Internet Feature Set](#) is present. Implemented for Mail only on OS version 3.0 or later.

---

#### **sysAppLaunchCmdAddRecord Parameter Block for Mail Application**

##### **Prototype**

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

typedef struct {
    Boolean secret;
    Boolean signature;
    Boolean confirmRead;
    Boolean confirmDelivery;
    MailMsgPriorityType priority;
    UInt8 padding
    Char* subject;
```

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

<b>Fields</b>		
	secret	True means that the message should be marked secret.
	signature	True means that the signature from the Mail application's preferences should be attached to the message.
	confirmRead	True means that a confirmation should be sent when the message is read.
	confirmDelivery	True means that a confirmation should be sent when the message is delivered.
	priority	Message priority. Specify one of the MailMsgPriorityType enumerated types.
	padding	Reserved for future use.
	subject	Message's subject, a null-terminated string (optional).
	from	Message's sender, a null-terminated string (not used on outgoing mail).
	to	Address of the recipient, a null-terminated string (required).
	cc	Addresses of recipients to be copied, a null-terminated string (optional).
	bcc	Addresses of recipients to be blind copied, a null-terminated string (optional).
	replyTo	Reply to address, a null-terminated string (optional).
	body	The text of the message, a null-terminated string (required).

## Application Launch Codes

### *Launch Codes*

---

#### **sysAppLaunchCmdAddRecord Parameter Block for iMessenger Application**

**Prototype**

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

<b>Fields</b>	<b>category</b>	Category in which to place the message. Specify one of the following categories:  MsgInboxCategory MsgOutboxCategory MsgDeletedCategory MsgFiledCategory MsgDraftCategory
	<b>edit</b>	True means that the message should be opened in the editor. False means that the message should simply be placed into the outbox and the editor not opened. You can specify true only if the category is set to MsgOutboxCategory.
	<b>signature</b>	True means that the signature from the iMessenger application preferences should be attached to the message.
	<b>subject</b>	Message's subject, a null-terminated string (optional).
	<b>from</b>	Message's sender, a null-terminated string (not used on outgoing mail).
	<b>to</b>	Address of the recipient, a null-terminated string (required).

replyTo	Reply to address, a null-terminated string (optional).
body	The text of the message, a null-terminated string (required).

## **sysAppLaunchCmdAlarmTriggered**

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

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

### **sysAppLaunchCmdAlarmTriggered Parameter Block**

#### **Prototype**

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

#### **Fields**

-> ref                 The caller-defined value specified when the alarm was set with [AlmSetAlarm](#).

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

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

## Application Launch Codes

### *Launch Codes*

---

padding      Not used.

## **sysAppLaunchCmdAttention**

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

```
typedef UInt16 AttnCommand;
```

The following table lists the values that AttnCommand can assume.

**Table 1.2 sysAppLaunchCmdAttention Commands**

Constant	Value	Description
AttnCommand_drawData il	((AttnCommand) 1)	Indicates that the application needs to draw the detailed contents of the attention dialog. The command arguments parameter points to a structure of type <a href="#"><u>AttnCommandDrawDetailArgsType</u></a> .
AttnCommand_drawList	((AttnCommand) 2)	Indicates that the application needs to draw the appropriate list item in the attention dialog. The command arguments parameter points to a structure of type <a href="#"><u>AttnCommandDrawListArgsType</u></a> .

**Table 1.2 sysAppLaunchCmdAttention Commands**

<b>Constant</b>	<b>Value</b>	<b>Description</b>
AttnCommand_customEffect	( (AttnCommand) 3 )	Indicates that the Attention Manager is doing something to get the user's attention, and any application-specific special effect should be done. This command is only sent to attention items that set the AttnFlags_CustomEffectBit when they call <a href="#">AttnGetAttention</a> , which most applications won't do.
AttnCommand_goThere	( (AttnCommand) 4 )	Tells the application to navigate to the item. The command arguments parameter is NULL. An application commonly calls <a href="#">SysAppLaunch</a> upon receipt of this command to have itself launched.
AttnCommand_gotIt	( (AttnCommand) 5 )	Tells the application that the user is dismissing the item. The command arguments parameter is NULL. The application may choose to clean up memory at this point.

## Application Launch Codes

### *Launch Codes*

---

**Table 1.2 sysAppLaunchCmdAttention Commands**

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

### **AttnCommandDrawDetailArgsType**

When AttnCommand\_drawDetail is passed to the application, either via the callback function or as a parameter accompanying the [sysAppLaunchCmdAttention](#) launch code, the application needs to draw the detailed contents of the attention dialog. The AttnCommandDrawDetailArgsType structure accompanies the AttnCommand\_drawDetail command, and provides the information needed to draw the contents of that dialog.

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

## Field Descriptions

bounds	Contains the window-relative bounding box for the area to draw. The clipping region is also set to the dimensions of this box to prevent accidental drawing outside.
firstTime	Set to true if the user has not yet seen this item. The value of this field could be used, for example, to display attentions that the user hasn't seen before in some special way.
flags	The global user preferences for this attention attempt combined with the custom flags passed in by the developer. For example, if the global preference is to mute sounds, and the developer flags are both zero, then the AttnFlags_NoSound flag is on and the AttnFlags_AlwaysSound flag is off.

## AttnCommandDrawListArgsType

When AttnCommand\_drawList is passed to the application, either via the callback function or as a parameter accompanying the [sysAppLaunchCmdAttention](#) launch code, the application is to draw the appropriate list item in the attention dialog. The AttnCommandDrawListArgsType structure accompanies the AttnCommand\_drawList command, and provides the information needed to draw the contents of that dialog.

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

## Application Launch Codes

### *Launch Codes*

---

```
} AttnCommandDrawListArgsType;
```

#### Field Descriptions

bounds	Contains the window-relative bounding box for the area to draw. The clipping region is also set to the dimensions of this box to prevent accidental drawing outside.
firstTime	Set to true if the user has not yet seen this item. The value of this field could be used, for example, to trigger a custom sound the first time this attention item is presented to the user.
flags	The global user preferences for this attention attempt combined with the custom flags passed in by the developer. For example, if the global preference is to mute sounds, and the developer flags were both zero, then the AttnFlags_NoSound flag is on and the AttnFlags_AlwaysSound flag is off.

#### AttnCommandGotItArgsType

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

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

### Field Descriptions

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

## sysAppLaunchCmdCardLaunch

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

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

### sysAppLaunchCmdCardLaunch Parameter Block

**Prototype**

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

## Application Launch Codes

### *Launch Codes*

---

```
} SysAppLaunchCmdCardType;
```

<b>Fields</b>		
	<- err	Initially set to expErrUnsupportedOperation, applications that recognize sysAppLaunchCmdCardLaunch and that don't want to receive the subsequent sysAppLaunchNormalLaunch launch code should set this field to errNone.
	-> volRefNum	The reference number of the volume from which the application is being launched.
	-> path	The complete path to the application being launched.
	<-> startFlags	This field is made up of a combination of the following flags.  sysAppLaunchStartFlagAutoStart Indicates that the application is being run automatically upon card insertion.  sysAppLaunchStartFlagNoUISwitch Set this bit to prevent a UI switch to the application.  sysAppLaunchStartFlagNoAutoDelete Set this bit to prevent the VFS Manager from deleting the copy of the application in main memory when the associated volume is unmounted.

## sysAppLaunchCmdCountryChange

Responds to country change.

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

## sysAppLaunchCmdDisplayAlarm

Performs full, possibly blocking, handling of alarm.

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

### **sysAppLaunchCmdDisplayAlarm Parameter Block**

#### **Prototype**

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

#### **Fields**

-> ref	The caller-defined value specified when the alarm was set with <a href="#">AlmSetAlarm</a> .
-> alarmSeconds	The date/time specified when the alarm was set with AlmSetAlarm. The value is given as the number of seconds since 1/1/1904.
-> soundAlarm	true if the alarm should be sounded, false otherwise. This value is currently not used.
padding	Not used.

### **sysAppLaunchCmdExgAskUser**

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

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

## Application Launch Codes

### *Launch Codes*

---

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

Starting with Palm OS 3.5, the Exchange Manager allows applications to provide extra functionality in the exchange dialog. You can have the dialog include a category pop-up list from which the user chooses a category in which to file the incoming data. If you want to provide a category pop-up list, call the [ExgDoDialog](#) function in response to this launch code and pass it a database that contains the categories to be listed. See the description of that function for more information.

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

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

<code>exgAskDialog</code>	Display the default exchange dialog provided by Exchange Manager.
<code>exgAskOk</code>	Accept the incoming data.
<code>exgAskCancel</code>	Reject the incoming data.

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

---

**IMPORTANT:** Implemented only if [3.0 New Feature Set](#) is present.

---

## **sysAppLaunchCmdExgAskUser Parameter Block**

<b>Prototype</b>	typedef struct { ExgSocketPtr           socketP; ExgAskResultType      result; UInt8                 reserved; } ExgAskParamType;
<b>Fields</b>	<-> socketP      Socket pointer (see <a href="#">ExgSocketType</a> ) <- result           Show dialog, auto-confirm, or auto-cancel -> reserved        Reserved for future use

## **sysAppLaunchCmdExgGetData**

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

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

The parameter block sent with this launch code is a pointer to the [ExgSocketType](#) structure corresponding to the Exchange Manager connection on which the data is to be sent. You pass this socket pointer to ExgPut. For more details, see the “[Exchange Manager](#)” chapter.

---

**IMPORTANT:** Implemented only if [4.0 New Feature Set](#) is present.

---

## **sysAppLaunchCmdExgPreview**

Following the launch code [sysAppLaunchCmdExgAskUser](#), the Exchange Manager sends the sysAppLaunchCmdExgPreview launch code to have the application display the preview in the exchange dialog.

## Application Launch Codes

### *Launch Codes*

---

#### **sysAppLaunchCmdExgPreview Parameter Block**

<b>Prototype</b>	<pre>typedef struct {     UInt16           version;     ExgSocketType   *socketP;     UInt16           op;     Char             *string;     UInt32           size;     RectangleType   bounds;     UInt16           types;     Err              error; } ExgPreviewInfoType;</pre>
<b>Fields</b>	
-> version	Set this field to 0 to specify version 0 of this structure.
-> socketP	A pointer to the socket structure (see <a href="#">ExgSocketType</a> ). The libraryRef field must point to the exchange library from which preview data should be received.
-> op	A constant that identifies the operation. This can be one of the following:
	<b>exgPreviewDialog</b> Display a form or modal dialog containing the preview. This constant is only used in situations where one application launches another to display data.
	<b>exgPreviewDraw</b> Draw the preview as a graphic in the bounds rectangle.
	<b>exgPreviewLongString</b> Return the preview as a long string in the string field.
	<b>exgPreviewQuery</b> Return the list of preview modes the application supports in the types field.
	<b>exgPreviewLongString</b> Return the preview as a short string in the string field.

<- string	A buffer into which the preview string is placed if one of the string preview operations is specified.
-> size	The allocated size of the <code>string</code> field.
-> bounds	The bounds of the rectangle in which to draw the graphic if the preview operation is <code>exgPreviewDraw</code> .
<- types	Upon return from <code>exgPreviewQuery</code> , a bit field identifying the types of previews the library supports ( <code>exgPreviewDraw</code> , <code>exgPreviewLongString</code> , or <code>exgPreviewShortString</code> ).
<- error	The error code returned from the library. If this is <code>errNone</code> , the preview operation was successful.

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

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

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

---

**IMPORTANT:** Implemented only if [4.0 New Feature Set](#) is present.

---

## Application Launch Codes

### Launch Codes

---

#### **sysAppLaunchCmdExgReceiveData**

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

The application should use Exchange Manager functions to receive the data and store it or do whatever it needs to with the data. Specifically, most applications should respond to this launch code by calling [ExgAccept](#) to accept the connection and then [ExgReceive](#) to receive the data.

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

---

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

---

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

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

---

**IMPORTANT:** Implemented only if [3.0 New Feature Set](#) is present.

---

#### **sysAppLaunchCmdFind**

This launch code is used to implement the global find. When the user enters a text string in the Find dialog, the system sends this launch code with the [FindParamsType](#) parameter block to each

application. The application should search for the string that the user entered and return any records matching the find request.

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

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

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

Applications that support this launch code should support [`sysAppLaunchCmdSaveData`](#) and [`sysAppLaunchCmdGoto`](#) as well.

### **sysAppLaunchCmdFind Parameter Block**

**Prototype**

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

## Application Launch Codes

### *Launch Codes*

---

```
    FindMatchType           match [maxFinds];  
} FindParamsType;
```

<b>Fields</b>	dbAccessMode	Mode in which to open the application's database. Pass this directly to <a href="#">DmOpenDatabase</a> as the mode parameter. Its value is either dmModeReadOnly or dmModeReadOnly   dmModeShowSecret. (See " <a href="#">Open Mode Constants</a> " for more information.)
	recordNum	Index of last record that contained a match. Start the search from this location. Do not set this value directly. Instead, call <a href="#">FindSaveMatch</a> when you have a matching record.
	more	If true, the Find results dialog displays the Find More button indicating that there may be more results to display.  Typically <a href="#">FindSaveMatch</a> handles setting the more field. Applications with large databases to search may want to periodically check for and stop the search if an event is pending. If so, they should set this field to true before stopping.
	strAsTyped	Search string as the user entered it.
	strToFind	Normalized version of the search string. The method by which a search string is normalized varies depending on the version of Palm OS and the character encoding supported by the device. You pass strToFind directly to the search function (either <a href="#">FindStrInStr</a> , <a href="#">TxtFindString</a> , or <a href="#">TxtGlueFindString</a> ).
	reserved1	Reserved for future use.
	numMatches	The current number of matches. Do not set this field directly. Instead, call <a href="#">FindSaveMatch</a> , which increments it for you.

lineNumber	Line number of the next line that displays the results. Do not set this field directly. It is incremented by a call to <a href="#">FindDrawHeader</a> .
continuation	If <code>true</code> , the launch code is being sent as part of a continuation of a previous Find. If <code>false</code> , this is a new Find. Do not set this field directly; the system sets it when the Find results dialog is full.
searchedCaller	If <code>true</code> , the application that was active at the time the user tapped the Find button has responded to this launch code. This application is always searched before any others.
callerAppDbID	Database ID of the application that was active when the user tapped the Find button. Do not change the value of this field; the system sets it and uses it when searching for application databases.
callerAppCardNo	Card number of the application that was active when the user tapped the Find button. Do not change the value of this field; the system sets it and uses it when searching for application databases.
appDbID	The ID of your application's resource database. Do not set this field directly; the system sets it and uses it when searching for application databases.
appCardNo	The card number of your application's resource database. Do not set this field directly; the system sets it and uses it when searching for application databases.
newSearch	System use only.
reserved2	Reserved for future use.
searchState	System use only.
match	System use only.

## Application Launch Codes

### *Launch Codes*

---

#### **sysAppLaunchCmdGoto**

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

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

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

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

For example:

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

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

#### **sysAppLaunchCmdGoto Parameter Block**

##### **Prototype**

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

---

<b>Fields</b>		
	searchStrLen	Length of normalized search string. This is <b>not</b> the length of the matching string. See below for a full explanation.
	dbCardNo	Card number of the database to open.
	dbID	Local ID of the database to open.
	recordNum	Index of record containing a match.
	matchPos	Position of the match within the field.
	matchFieldNum	Field number string was found in.
	matchCustom	Application-specific information.

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

## **sysAppLaunchCmdGoToURL**

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

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

This launch code may be sent in the following instances:

- If the [Wireless Internet Feature Set](#) is, applications can send this launch code directly to the Web Clipping Application Viewer application.
- If [4.0 New Feature Set](#) is present, the [ExgRequest](#) function launches an application with this launch code if it cannot find an exchange library that is registered for the URL it has

## **Application Launch Codes**

### *Launch Codes*

---

received. To receive the launch code, the application must first use [ExgRegisterDatatype](#) to register for a URL scheme.

---

**IMPORTANT:** Implemented only if [Wireless Internet Feature Set](#) is present.

---

### **sysAppLaunchCmdHandleSyncCallApp**

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

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

**sysAppLaunchCmdHandleSyncCallApp Parameter Block****Prototype**

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

**Fields**

pbSize	Size, in bytes, of this parameter block. Set to sizeof(SysAppLaunchCmdHandleSyncCallAppType).
action	Call action ID (application-specific).
paramP	Pointer to parameter block (call action ID specific).
dwParamSize	Parameter block size, in bytes.
dlRefP	DesktopLink reference pointer. Supply this value in the <a href="#">DlkCallAppReplyParamType</a> structure when calling <a href="#">DlkControl</a> with the dlkCtlSendCallAppReply control code.
handled	Initialized to false by DLServer; if handled, your application must set it to true (and your handler the handler must call DlkControl with the dlkCtlSendCallAppReplycontrol code). If your handler is not going to send a reply back to the conduit, leave this field set to false, in which case the DesktopLink Server will send the default "unknown request" reply.
reserved1	Reserved. Set to NULL.

## Application Launch Codes

### *Launch Codes*

---

replyErr	Error code returned from the call to <a href="#">DlkControl</a> with the dlkCtlSendCallAppReply control code.
dwReserved1	Reserved. Set to NULL.
dwReserved2	Reserved. Set to NULL.

## **sysAppLaunchCmdInitDatabase**

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

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

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

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

### **sysAppLaunchCmdInitDatabase Parameter Block**

**Prototype**

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

**Fields**

dbP	Database reference.
creator	Database creator.
type	Database type.

version Database version.

## **sysAppLaunchCmdLookup**

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

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

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

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

### **sysAppLaunchCmdLookup Parameter Block**

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

---

**IMPORTANT:** Implemented only if [2.0 New Feature Set](#) is present.

---

## **sysAppLaunchCmdNotify**

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

---

**IMPORTANT:** Implemented only if [Notification Feature Set](#) is present.

---

## Application Launch Codes

### *Launch Codes*

---

#### **sysAppLaunchCmdNotify Parameter Block**

The [SysNotifyParamType](#) structure declared in `NotifyMgr.h` defines the format of this launch code's parameter block. See its description in the “Notifications” chapter.

#### **sysAppLaunchCmdOpenDB**

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

---

**IMPORTANT:** Implemented only if [Wireless Internet Feature Set](#) is present.

---

#### **sysAppLaunchCmdOpenDB Parameter Block**

**Prototype**

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

**Fields**

cardNo	Card number of database to open.
dbID	Database id of database to open.

#### **sysAppLaunchCmdPanelCalledFromApp**

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

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

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

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

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

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

---

**IMPORTANT:** Implemented only if [2.0 New Feature Set](#) is present.

---

## **sysAppLaunchCmdReturnFromPanel**

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

---

**IMPORTANT:** Implemented only if [2.0 New Feature Set](#) is present.

---

## **sysAppLaunchCmdSaveData**

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

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

## Application Launch Codes

### *Launch Codes*

---

#### **sysAppLaunchCmdSaveData Parameter Block**

<b>Prototype</b>	<pre>typedef struct {     Boolean      uiComing;     UInt8       reserved1; } SysAppLaunchCmdSaveDataType;</pre>				
<b>Fields</b>	<table><tr><td>uiComing</td><td>true if the system dialog is displayed before launch code arrives.</td></tr><tr><td>reserved1</td><td>Reserved for future use.</td></tr></table>	uiComing	true if the system dialog is displayed before launch code arrives.	reserved1	Reserved for future use.
uiComing	true if the system dialog is displayed before launch code arrives.				
reserved1	Reserved for future use.				

#### **sysAppLaunchCmdSyncNotify**

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

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

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

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

#### **sysAppLaunchCmdSystemLock**

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

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

---

**IMPORTANT:** Implemented only if [2.0 New Feature Set](#) is present.

---

## **sysAppLaunchCmdSystemReset**

Launch code to respond to system soft or hard reset.

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

### **sysAppLaunchCmdSystemReset Parameter Block**

**Prototype**

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

**Fields**

hardReset	true if system was hardReset. false if system was softReset.
createDefaultDB	If true, application has to create default database.

## **sysAppLaunchCmdTimeChange**

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

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

On systems that have the [Notification Feature Set](#), applications should register to receive the `sysNotifyTimeChangeEvent`

## Application Launch Codes

### *Launch Flags*

---

notification instead of responding to this launch code. The sysAppLaunchCmdTimeChange launch code is sent to all applications. The sysNotifyTimeChangeEvent notification is sent only to applications that have specifically registered to receive it, making it more efficient than sysAppLaunchCmdTimeChange.

### **sysAppLaunchCmdURLParams**

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

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

An application launched with this code may or may not have access to global variables, static local variables, and code segments other than segment 0 (in multi-segment applications). It depends on the URL that caused the Web Clipping Application Viewer to send this launch code. If this launch code results from a `palm` URL, then globals are available. If the launch code results from a `palmcall` URL, then globals are not available.

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

---

**IMPORTANT:** Implemented only if [Wireless Internet Feature Set](#) is present.

---

## Launch Flags

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

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

**Table 1.3 Launch Flags**

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

---

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

---

## **Application Launch Codes**

### *Launch Flags*

---

# Palm OS Events

---

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

Event	UI Object
<a href="#">appStopEvent</a>	N.A.
<a href="#">ctlEnterEvent</a> , <a href="#">ctlExitEvent</a> , <a href="#">ctlRepeatEvent</a> , <a href="#">ctlSelectEvent</a>	Control
<a href="#">daySelectEvent</a>	N.A.
<a href="#">fldChangedEvent</a> , <a href="#">fldEnterEvent</a> , <a href="#">fldHeightChangedEvent</a>	Field
<a href="#">frmCloseEvent</a> , <a href="#">frmGotoEvent</a> , <a href="#">frmLoadEvent</a> , <a href="#">frmOpenEvent</a> , <a href="#">frmSaveEvent</a> , <a href="#">frmUpdateEvent</a> , <a href="#">frmTitleEnterEvent</a> , <a href="#">frmTitleSelectEvent</a>	Form
<a href="#">frmGadgetEnterEvent</a> , <a href="#">frmGadgetMiscEvent</a>	Extended gadget
<a href="#">inetSockReadyEvent</a> , <a href="#">inetSockStatusChangeEvent</a>	N.A. (INetLib)
<a href="#">keyDownEvent</a>	N.A.
<a href="#">lstEnterEvent</a> , <a href="#">lstExitEvent</a> , <a href="#">lstSelectEvent</a>	List
<a href="#">menuEvent</a> , <a href="#">menuOpenEvent</a> , <a href="#">menuCloseEvent</a> , <a href="#">menuCmdBarOpenEvent</a>	Menu
<a href="#">nilEvent</a>	N.A.
<a href="#">penDownEvent</a> , <a href="#">penMoveEvent</a> , <a href="#">penUpEvent</a>	N.A. (pen)

## Palm OS Events

### *Event Data Structures*

---

Event	UI Object
<a href="#"><u>popSelectEvent</u></a>	Popup (Control)
<a href="#"><u>sclEnterEvent</u></a> , <a href="#"><u>sclRepeatEvent</u></a> , <a href="#"><u>sclExitEvent</u></a>	Scroll bar
<a href="#"><u>tblEnterEvent</u></a> , <a href="#"><u>tblExitEvent</u></a> , <a href="#"><u>tblSelectEvent</u></a>	Table
<a href="#"><u>winEnterEvent</u></a> , <a href="#"><u>winExitEvent</u></a>	Window

## Event Data Structures

### **eventsEnum**

The eventsEnum enum specifies the possible event types.

```
enum events {
    nilEvent = 0,
    penDownEvent,
    penUpEvent,
    penMoveEvent,
    keyDownEvent,
    winEnterEvent,
    winExitEvent,
    ctlEnterEvent,
    ctlExitEvent,
    ctlSelectEvent,
    ctlRepeatEvent,
    lstEnterEvent,
    lstSelectEvent,
    lstExitEvent,
    popSelectEvent,
    fldEnterEvent,
    fldHeightChangedEvent,
    fldChangedEvent,
    tblEnterEvent,
    tblSelectEvent,
    daySelectEvent,
    menuEvent,
    appStopEvent = 22,
    frmLoadEvent,
```

```
frmOpenEvent,
frmGotoEvent,
frmUpdateEvent,
frmSaveEvent,
frmCloseEvent,
frmTitleEnterEvent,
frmTitleSelectEvent,
tblExitEvent,
sclEnterEvent,
sclExitEvent,
sclRepeatEvent,
tsmFepModeEvent,

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

firstINetLibEvent = 0x1000,
firstWebLibEvent = 0x1100,

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

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

## **EventType**

The EventType structure contains all the data associated with a system event. All event types have some common data. Most events also have data specific to those events. The specific data uses a union that is part of the EventType data structure. The union can have up to 8 words of specific data.

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

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

### Common Field Descriptions

eType	One of the <a href="#">eventsEnum</a> constants. Specifies the type of the event.
penDown	true if the pen was down at the time of the event, otherwise false.
tapCount	The number of taps received at this location. This value is used mainly by fields. When the user taps in a text field, two taps selects a word, and three taps selects the entire line.
screenX	Window-relative position of the pen in pixels (number of pixels from the left bound of the window).
screenY	Window-relative position of the pen in pixels (number of pixels from the top left of the window).
data	The specific data for an event, if any. The data is a union, and its exact contents depend on the eType field. The <a href="#">Event Reference</a> section in this chapter shows what the data field contains for each event.

---

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

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

---

## Compatibility

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

## EventPtr

The `EventPtr` defines a pointer to an [`EventType`](#).

```
typedef EventType *EventPtr;
```

# Event Reference

## appStopEvent

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

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

## ctlEnterEvent

The control routine [CtlHandleEvent](#) sends this event when it receives a [penDownEvent](#) within the bounds of a control.

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

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

### Field Descriptions

**controlID** Developer-defined ID of the control.

**pControl** Pointer to a control structure ([ControlType](#)).

## ctlExitEvent

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

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

### Field Descriptions

**controlID** Developer-defined ID of the control.

**pControl** Pointer to a control structure ([ControlType](#)).

## ctlRepeatEvent

The control routine [CtlHandleEvent](#) sends this event. When CtlHandleEvent receives a [ctlEnterEvent](#) in a repeating

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

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

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

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

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

### Field Descriptions

`controlID`      Developer-defined ID of the control.

`pControl`      Pointer to a control structure ([ControlType](#)).

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

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

**Compatibility**      The `value` field is only present if [3.5 New Feature Set](#) is present.

## ctlSelectEvent

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

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

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

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

### Field Descriptions

<code>controlID</code>	Developer-defined ID of the control.
<code>pControl</code>	Pointer to a control structure ( <a href="#">ControlType</a> ).
<code>on</code>	<code>true</code> when the control is depressed; otherwise, <code>false</code> .
<code>reserved1</code>	Unused.
<code>value</code>	Current value if the control is a slider.

**Compatibility** The `value` field is only present if [3.5 New Feature Set](#) is present.

## daySelectEvent

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

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

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

```
struct daySelect {  
    struct DaySelectorType *pSelector;
```

```
    Int16 selection;
    Boolean useThisDate;
    UInt8 reserved1;
} daySelect;
```

### **Field Descriptions**

pSelector	Pointer to a day selector structure (DaySelectorType).
selection	Not used.
useThisDate	Set to true to automatically use the selected date.
reserved1	Unused.

## **fldChangedEvent**

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

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

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

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

### **Field Descriptions**

fieldID	Developer-defined ID of the field.
pField	Pointer to a field structure (FieldType).

## **fldEnterEvent**

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

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

### Field Descriptions

fieldID    Developer-defined ID of the field.

pField    Pointer to a field structure ([FieldType](#)).

## fldHeightChangedEvent

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

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

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

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

### Field Descriptions

fieldID    Developer-defined ID of the field.

pField    Pointer to a field structure ([FieldType](#)).

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

currentPos    Current position of the insertion point.

## **frmCloseEvent**

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

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

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

### **Field Descriptions**

formID      Developer-defined ID of the form.

## **frmGadgetEnterEvent**

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

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

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

### **Field Descriptions**

gadgetID      Developer-defined ID of the gadget.

gadgetP      Pointer to the [FormGadgetType](#) object representing this gadget.

**Compatibility**      Implemented only if [3.5 New Feature Set](#) is present.

## **frmGadgetMiscEvent**

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

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

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

### **Field Descriptions**

gadgetID	Developer-defined ID of the gadget.
gadgetP	Pointer to the <a href="#">FormGadgetType</a> object representing this gadget.
selector	Any necessary integer value to pass to the gadget handler function.
dataP	A pointer to any necessary data to pass to the gadget handler function.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

## **frmGotoEvent**

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

The application is responsible for handling this event.

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

```
struct frmGoto {  
    UInt16 formID;  
    UInt16 recordNum;  
    UInt16 matchPos;  
    UInt16 matchLen;  
    UInt16 matchFieldNum;
```

```
    UInt32 matchCustom;
} frmGoto;
```

### Field Descriptions

formID	Developer-defined ID of the form.
recordNum	Index of record containing the match string.
matchPos	Position of the match.
matchLen	Length of the matched string.
matchFieldNum	Number of the field the matched string was found in.
matchCustom	Application-specific information. You might use this if you need to provide extra information to locate the matching string within the record.

## frmLoadEvent

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

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

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

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

### Field Descriptions

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

## frmOpenEvent

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

The application is responsible for handling this event.

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

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

### **Field Descriptions**

**formID**      Developer-defined ID of the form.

## **frmSaveEvent**

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

The application is responsible for handling this event.

No data is passed with this event.

## **frmTitleEnterEvent**

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

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

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

### **Field Descriptions**

**formID**      Developer-defined ID of the form.

## **frmTitleSelectEvent**

The control routine [FrmHandleEvent](#) sends this event.

[FrmHandleEvent](#) receives a [frmTitleEnterEvent](#), it tracks the pen until the pen is lifted. If the pen is lifted within the bounds of the active same title bar region, a [frmTitleSelectEvent](#) is added to the event queue.

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

## Palm OS Events

### Event Reference

---

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

#### Field Descriptions

**formID** Developer-defined ID of the form.

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

## frmUpdateEvent

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

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

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

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

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

### Field Descriptions

- formID      Developer-defined ID of the form.
- updateCode    The reason for the update request. FrmEraseForm sets this code to frmRedrawUpdateCode, which indicates that the entire form needs to be redrawn. Application developers can define their own updateCode. The updateCode is passed as a parameter to [FrmUpdateForm](#).

## inetSockReadyEvent

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

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

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

### Field Descriptions

- sockH        Socket handle of the socket that this event refers to.
- context      Not used.
- inputReady    true when the socket has data ready for the [INetLibSockRead](#) call.
- outputReady   Not used.

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

**Compatibility**   Implemented only if [Wireless Internet Feature Set](#) is present.

## inetSockStatusChangeEvent

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

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

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

### Field Descriptions

sockH	Socket handle of the socket that this event refers to.
context	Not used.
status	Current status of the socket. This is one of the <a href="#">INetStatusEnum</a> constants.
sockErr	Reason for failure of the last operation, if any. The current socket error can be cleared by calling <a href="#">INetLibSockStatus</a> .

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

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

## keyDownEvent

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

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

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

### **Field Descriptions**

chr	The character code.
keyCode	Unused.
modifiers	0, or one or more of the following values:
shiftKeyMask	Graffiti is in case-shift mode.
capsLockMask	Graffiti is in cap-shift mode.
numLockMask	Graffiti is in numeric-shift mode.
commandKeyMask	The Graffiti glyph was the menu command glyph or a virtual key code.
optionKeyMask	Not implemented. Reserved.
controlKeyMask	Not implemented. Reserved.
autoRepeatKeyMask	Event was generated due to auto-repeat.
doubleTapKeyMask	Not implemented. Reserved.
poweredOnKeyMask	The key press caused the system to be powered on.
appEvtHookKeyMask	System use only.
libEvtHookKeyMask	System use only.

### **LstEnterEvent**

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

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

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

#### Field Descriptions

listID      Developer-defined ID of the list.  
pList        Pointer to a list structure ([ListType](#)).  
selection    Unused.

## IstExitEvent

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

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

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

#### Field Descriptions

listID      Developer-defined ID of the list.  
pList        Pointer to a list structure ([ListType](#)).

## IstSelectEvent

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

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

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

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

### **Field Descriptions**

<code>listID</code>	Developer-defined ID of the list.
<code>pList</code>	Pointer to a list structure ( <a href="#">ListType</a> ).
<code>selection</code>	Item number (zero-based) of the new selection.

## **menuCloseEvent**

This event is not currently used.

## **menuCmdBarOpenEvent**

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

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

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

### Field Descriptions

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

`reserved` Unused.

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

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

## menuEvent

The menu routine [MenuHandleEvent](#) sends this event:

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

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

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

### Field Descriptions

`itemID` Item ID of the selected menu command.

## menuOpenEvent

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

A menu remains active until one of the following happens:

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

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

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

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

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

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

### Field Descriptions

`menuRscID` Resource ID of the menu.

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

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

## **nilEvent**

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

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

## **penDownEvent**

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

The following data is passed with the event:

### **Field Descriptions**

penDown Always true.

tapCount The number of taps received at this location.

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

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

## **penMoveEvent**

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

The following data is passed with the event:

### **Field Descriptions**

penDown Always true.

tapCount The number of taps received at this location.

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

## **penUpEvent**

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

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

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

### **Field Descriptions**

start      Display-relative start point of the stroke.

end      Display-relative end point of the stroke.

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

penDown      Always false.

tapCount      The number of taps received at this location.

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

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

## **popSelectEvent**

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

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

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

### Field Descriptions

controlID	Developer-defined ID of the resource.
controlP	Pointer to the control structure ( <a href="#">ControlType</a> ) of the popup trigger object.
listID	Developer-defined ID of the popup list object.
listP	Pointer to the list structure ( <a href="#">ListType</a> ) of the popup list object.
selection	Item number (zero-based) of the new list selection.
priorSelection	Item number (zero-based) of the prior list selection.

## sclEnterEvent

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

Applications usually don't have to handle this event.

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

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

### **Field Descriptions**

scrollBarID      Developer-defined ID of the scroll bar resource.  
pScrollBar      Pointer to the scroll bar structure.

## **sclExitEvent**

The routine [SclHandleEvent](#) sends this event when the user lifts the pen from the scroll bar.

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

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

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

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

### **Field Descriptions**

scrollBarID      Developer-defined ID of the scroll bar resource.  
pScrollBar      Pointer to the scroll bar structure.  
value              Initial position of the scroll bar  
newValue          New position of the scroll bar. Given value and newValue, you can actually tell how much you have scrolled.

## **sclRepeatEvent**

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

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

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

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

### Field Descriptions

scrollBarID	Developer-defined ID of the scroll bar resource.
pScrollBar	Pointer to the scroll bar structure.
value	Initial position of the scroll bar.
newValue	New position of the scroll bar. Given value and newValue, you can actually tell how much you have scrolled.
time	System-ticks count when the event is added to the queue to determine when the next event should occur.

## tblEnterEvent

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

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

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

### **Field Descriptions**

tableID	Developer-defined ID of the table.
pTable	Pointer to a table structure ( <a href="#">TableType</a> ).
row	Row of the item.
column	Column of the item.

## **tblExitEvent**

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

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

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

### **Field Descriptions**

tableID	Developer-defined ID of the table.
pTable	Pointer to a table structure ( <a href="#">TableType</a> ).
row	Row of the item.
column	Column of the item.

## **tblSelectEvent**

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

added to the event queue; if not, a [tblExitEvent](#) is added to the event queue.

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

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

#### Field Descriptions

tableID Developer-defined ID of the table.

pTable Pointer to a table structure ([TableType](#)).

row Row of the item.

column Column of the item.

## winEnterEvent

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

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

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

### **Field Descriptions**

enterWindow Handle to the window we are entering. If the window is a form, then this is a pointer to a [FormType](#) structure; if not, it's a pointer to a [WindowType](#) structure.

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

## **winExitEvent**

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

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

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

### **Field Descriptions**

enterWindow Handle to the window we are entering. If the window is a form, then this is a pointer to a [FormType](#) structure; if not, it's a pointer to a [WindowType](#) structure.

exitWindow Handle to the window we are exiting. If the window is a form, then this is a pointer to a [FormType](#) structure; if not, it's a pointer to a [WindowType](#) structure.

## Palm OS Events

### *Event Reference*

---

# Notifications

---

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

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

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

See the “[Notification Manager](#)” chapter in this book and the section “[Notifications](#)” on page 30 of the *Palm OS Programmer’s Companion*, vol. I for more information on receiving and handling notifications.

**Table 3.1 Notification Constants**

Constant	Description
<a href="#"><u>cncNotifyProfileEvent</u></a>	The connection profile used by the Connection Panel has changed.
<a href="#"><u>sysExternalConnectorAttachEvent</u></a>	A device has been attached to an external connector.
<a href="#"><u>sysExternalConnectorDetachEvent</u></a>	A device has been detached from an external connector.
<a href="#"><u>sysNotifyAntennaRaisedEvent</u></a>	The antenna has been raised on a Palm VII™ series device.
<a href="#"><u>sysNotifyAppLaunchingEvent</u></a>	An application is about to be launched.
<a href="#"><u>sysNotifyAppQuittingEvent</u></a>	An application has just quit.

## Notifications

---

**Table 3.1 Notification Constants (*continued*)**

Constant	Description
<a href="#"><u>sysNotifyCardInsertedEvent</u></a>	An expansion card has been inserted into the expansion slot.
<a href="#"><u>sysNotifyCardRemovedEvent</u></a>	An expansion card has been removed from the expansion slot.
<a href="#"><u>sysNotifyDBCreatedEvent</u></a>	A database has been created.
<a href="#"><u>sysNotifyDBChangedEvent</u></a>	Database info has been set on a database, such as with <a href="#"><u>DmSetDatabaseInfo</u></a> .
<a href="#"><u>sysNotifyDBDeletedEvent</u></a>	A database has been deleted.
<a href="#"><u>sysNotifyDBDirtyEvent</u></a>	A database has been opened for write or in some other way has been made modifiable.
<a href="#"><u>sysNotifyDeleteProtectedEvent</u></a>	The Launcher has attempted to delete a protected database.
<a href="#"><u>sysNotifyDeviceUnlocked</u></a>	The user has unlocked the device.
<a href="#"><u>sysNotifyDisplayChangeEvent</u></a>	The color table or bit depth has changed.
<a href="#"><u>sysNotifyEarlyWakeupEvent</u></a>	The system is starting to wake up.
<a href="#"><u>sysNotifyEventDequeuedEvent</u></a>	An event has been removed from the event queue with <code>EvtGetEvent</code> .
<a href="#"><u>sysNotifyForgotPasswordEvent</u></a>	The user has tapped the Lost Password button in the Security application.
<a href="#"><u>sysNotifyGotUsersAttention</u></a>	The Attention Manager has informed the user of an event.
<a href="#"><u>sysNotifyHelperEvent</u></a>	An application has requested that a particular service be performed.
<a href="#"><u>sysNotifyIdleTimeEvent</u></a>	The system is idle and is about to doze.

**Table 3.1 Notification Constants (*continued*)**

<b>Constant</b>	<b>Description</b>
<u><a href="#">sysNotifyInsPtEnableEvent</a></u>	The insertion point is being enabled or disabled.
<u><a href="#">sysNotifyIrDASniffEvent</a></u>	Not used.
<u><a href="#">sysNotifyKeyboardDialogEvent</a></u>	The keyboard dialog is about to be displayed.
<u><a href="#">sysNotifyLateWakeUpEvent</a></u>	The system has finished waking up.
<u><a href="#">sysNotifyLocaleChangedEvent</a></u>	The system locale has changed.
<u><a href="#">sysNotifyMenuBarOpenEvent</a></u>	The system is about to display the menu command toolbar.
<u><a href="#">sysNotifyNetLibIFMediaEvent</a></u>	The system has been connected to or disconnected from the network.
<u><a href="#">sysNotifyPhoneEvent</a></u>	Reserved for future use.
<u><a href="#">sysNotifyPOSEMountEvent</a></u>	System use only.
<u><a href="#">sysNotifyProcessPenStrokeEvent</a></u>	The user has made a pen stroke on the silkscreen portion of the digitizer.
<u><a href="#">sysNotifyResetFinishedEvent</a></u>	The system has finished a reset.
<u><a href="#">sysNotifyRetryEnqueueKey</a></u>	The Attention Manager has failed to post a virtual character to the key queue.
<u><a href="#">sysNotifySleepNotifyEvent</a></u>	The system is about to go to sleep.
<u><a href="#">sysNotifySleepRequestEvent</a></u>	The system has decided to go to sleep.
<u><a href="#">sysNotifySyncFinishEvent</a></u>	A HotSync® operation has just completed.
<u><a href="#">sysNotifySyncStartEvent</a></u>	A HotSync operation is about to begin.
<u><a href="#">sysNotifyTimeChangeEvent</a></u>	The system time has just changed.
<u><a href="#">sysNotifyVirtualCharHandlingEvent</a></u>	A virtual character is being handled.

## Notifications

### *Notification Data Structures*

---

**Table 3.1 Notification Constants (*continued*)**

Constant	Description
<a href="#">sysNotifyVolumeMountedEvent</a>	A file system has been mounted.
<a href="#">sysNotifyVolumeUnmountedEvent</a>	A file system has been unmounted.

## Notification Data Structures

### SysNotifyParamType

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

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

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

## Field Descriptions

notifyType	The type of event that occurred. See <a href="#">Notification Reference</a> .
broadcaster	The creator ID of the application that broadcast the notification, or <code>sysNotifyBroadcasterCode</code> if the system broadcast the event.
notifyDetailsP	Pointer to data specific to this notification.
userDataP	Custom data that your notification handler requires. You create this data and pass it to <a href="#">SysNotifyRegister</a> .
handled	Set to <code>true</code> if the notification has been handled; set to <code>false</code> otherwise. In some cases, <code>handled</code> is treated as a bit field that notification handlers can use to indicate that certain conditions are true.
reserved2	Reserved for future use.

# Notification Reference

## cncNotifyProfileEvent

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

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

### cncNotifyProfileEvent Specific Data

`notifyDetailsP` points to a `CncProfileNotifyDetailsType` structure.

## Notifications

### Notification Reference

---

#### Prototype

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

#### Fields

version	The current version of this structure. Use the <code>kCncProfileNotifyCurrentVersion</code> constant to find out what the current version is.
profileID	The ID of the modified connection profile.
deviceKind	Device kind of the profile. This can be one of the following constants:  <code>kCncDeviceKindSerial</code> Serial connection profile.  <code>kCncDeviceKindModem</code> Modem profile.  <code>kCncDeviceKindPhone</code> Phone profile.  <code>kCncDeviceKindLocalNetwork</code> LAN profile.
request	The action that was performed. This can be one of the following constants:  <code>kCncNotifyCreateRequest</code> The profile has been created.  <code>kCncNotifyDeleteRequest</code> The profile is about to be deleted.  <code>kCncNotifyModifyRequest</code> The profile has been modified.

**kCncNotifyUpdateListRequest**

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

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

**kCncBecomeCurrentModifier**

The new profile should be made the current profile.

**kCncNotifyAlertUserModifier**

The user is prompted to set the current profile.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## **sysExternalConnectorAttachEvent**

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

### **sysExternalConnectorAttachEvent Specific Data**

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

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## **sysExternalConnectorDetachEvent**

The sysExternalConnectorDetachEvent is broadcast when a USB cradle, a RS-232 cradle or peripheral, a power cable, or a modem is detached from the universal connector. This notification is only broadcast on devices that have the universal connector.

## Notifications

### Notification Reference

---

#### **sysExternalConnectorDetachEvent Specific Data**

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

<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
----------------------	---

#### **sysNotifyAntennaRaisedEvent**

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

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

#### **sysNotifyAntennaRaisedEvent Specific Data**

None.

<b>Compatibility</b>	Implemented only if <a href="#">Notification Feature Set</a> is present.
----------------------	--

#### **sysNotifyAppLaunchingEvent**

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

#### **sysNotifyAppLaunchingEvent Specific Data**

notifyDetailsP points to a SysNotifyAppLaunchOrQuitType structure.

**Prototype**

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

<b>Fields</b>	version	The current version of this structure. The current version is 0.
---------------	---------	--

dbID	The local ID of the application.
cardNo	The number of the card on which the application resides.

**Compatibility** This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don't broadcast this notification. Palm OS 5 does broadcast it. Later versions may or may not broadcast this notification.

## **sysNotifyAppQuittingEvent**

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

### **sysNotifyAppLaunchingEvent Specific Data**

`notifyDetailsP` points to a `SysNotifyAppLaunchOrQuitType` structure. See the description of [sysNotifyAppLaunchingEvent](#) for a description of this structure.

**Compatibility** This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don't broadcast this notification. Palm OS 5 does broadcast it. Later versions Palm OS 5 does broadcast it. Later versions may or may not broadcast this notification.

## **sysNotifyCardInsertedEvent**

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

Most applications will want to register for [sysNotifyVolumeMountedEvent](#) instead of this notification. Register for `sysNotifyCardInsertedEvent` if you need to know when a card is inserted or if you want to prevent the Expansion Manager from performing its default handling of the notification.

## Notifications

### Notification Reference

---

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

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

#### **sysNotifyCardInsertedEvent Specific Data**

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

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## **sysNotifyCardRemovedEvent**

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

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

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

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

#### **sysNotifyCardRemovedEvent Specific Data**

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

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.



## sysNotifyDBCreatedEvent

The sysNotifyDBCreatedEvent is broadcast sometime *after* a database is created with [DmCreateDatabase](#).

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

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

### sysNotifyDBCreatedEvent Specific Data

notifyDetailsP points to a SysNotifyDBCreatedType structure.

**Prototype**

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

<b>Fields</b>		
dbName		Database name.
creator		Database creator ID.
type		Database type.
newDBID		Local ID of the newly-created database.

## Notifications

### Notification Reference

---

cardno	Card number upon which the database resides.
resDB	true if the database is a resource database, false otherwise.
padding	Structure padding byte.

**Compatibility** Implemented only if [5.0 New Feature Set](#) is present.



## sysNotifyDBChangedEvent

The sysNotifyDBChangedEvent is broadcast sometime *after* database info is set with [DmSetDatabaseInfo](#).

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

---

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

---

### sysNotifyDBChangeEvent Specific Data

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

**Prototype**

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

<b>Fields</b>		
dbName		New name of database.
dbID		Database ID.
creator		New database creator ID.
type		New database type.
crDate		New database creation date.
modDate		New database modification date.
bckUpDate		New database backup date.
modNum		New database modification number.
appInfoID		New database application info block.
sortInfoID		New database sort info block.
attributes		New database attributes.
cardNo		Card number upon which the dabatase resides.

## Notifications

### Notification Reference

---

version	New database version.
fields	Flags that indicate what about the database changed, and thus which of the above fields are set. The constants that define the <code>fields</code> bits are:
Flag	Value
DBChangedFieldSetName	0x1
DBChangedFieldSetCreator	0x2
DBChangedFieldSetType	0x4
DBChangedFieldSetCrDate	0x8
DBChangedFieldSetModDate	0x10
DBChangedFieldSetBckUpDate	0x20
DBChangedFieldSetModNum	0x40
DBChangedFieldSetAppInfo	0x80
DBChangedFieldSetSortInfo	0x100
DBChangedFieldSetAttributes	0x200
DBChangedFieldSetVersion	0x400
oldName	Name of database prior to the call to <code>DmSetDatabaseInfo</code> .
oldCreator	Database creator ID prior to the call to <code>DmSetDatabaseInfo</code> .
oldType	Database type prior to the call to <code>DmSetDatabaseInfo</code> .
oldAttributes	Database attributes prior to the call to <code>DmSetDatabaseInfo</code> .
padding	Structure padding bytes.

**Compatibility** Implemented only if [5.0 New Feature Set](#) is present.

## **sysNotifyDBDeletedEvent**

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

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

---

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

---

### **sysNotifyDBDeletedEvent Specific Data**

notifyDetailsP points to a SysNotifyDBDeletedType structure.

#### **Prototype**

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

#### **Fields**

oldDBID

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

---

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

---

## Notifications

### Notification Reference

---

cardNo	The number of the card on which the database resided.
attributes	The deleted database's attributes.
dbName	The name of the deleted database.
creator	The creator ID of the deleted database.
type	The type of the deleted database.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.



## sysNotifyDBDirtyEvent

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

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

---

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

---

### sysNotifyDBDirtyEvent Specific Data

notifyDetailsP points to a SysNotifyDBDirtyType structure.

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

<b>Fields</b>	<code>dbName</code>	Database name.
	<code>creator</code>	Database creator ID.
	<code>type</code>	Database type.

**Compatibility**    Implemented only if [5.0 New Feature Set](#) is present.

## **sysNotifyDeleteProtectedEvent**

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

Register for this notification if you have a protected database but you still want to allow users to delete your application or other code resource if they choose. A notification handler should check the information in the `notifyDetailsP` struct to see if its database is the one being deleted. If so, it should respond to this notification to perform any necessary cleanup and to clear the protected flag. In this way, when the Launcher attempts to delete the database again, it will succeed. Note that if an application has multiple protected databases, this notification may be sent out more than once.

### **sysNotifyDeleteProtectedEvent Specific Data**

`notifyDetailsP` points to a `SysNotifyDBInfoType` structure.

**Prototype**    `typedef struct SysNotifyDBInfoTag {  
 LocalID dbID;  
 UInt16 cardNo;  
 UInt16 attributes;  
 Char dbName [dmDBNameLength];  
 UInt32 creator;`

## Notifications

### Notification Reference

---

```
    UInt32 type;
} SysNotifyDBInfoType;
```

<b>Fields</b>	dbID	The local ID of the database to be deleted.
	cardNo	The number of the card on which the database resides.
	attributes	The database's attributes.
	dbName	The name of the database to be deleted.
	creator	The creator ID of the database to be deleted.
	type	The type of the database to be deleted.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## sysNotifyDeviceUnlocked

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

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

### sysNotifyDeviceUnlocked Specific Data

None.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## sysNotifyDisplayChangeEvent

The sysNotifyDisplayChangeEvent is broadcast whenever the display mode changes. That is, either the color table has been set to

use a specific palette using the [WinPalette](#) function or the bit depth has changed using the [WinScreenMode](#) function.

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

### **sysNotifyDisplayChangeEvent Specific Data**

`notifyDetailsP` points to a `SysNotifyDisplayChangeDetailsType` structure.

#### **Prototype**

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

#### **Fields**

<code>oldDepth</code>	The old bit depth.
<code>newDepth</code>	The new bit depth.

#### **Compatibility**

Implemented only if [Notification Feature Set](#) is present.

## **sysNotifyEarlyWakeupsEvent**

The `sysNotifyEarlyWakeupsEvent` is broadcast during [SysHandleEvent](#) immediately after the system has finished sleeping. The screen may still be turned off, and the system may not fully wake up. It may simply handle an alarm or a battery charger event and go back to sleep. Most applications that need notification of a wakeup event will probably want to register for [sysNotifyLateWakeupsEvent](#) instead.

---

**IMPORTANT:** This notification is **not** guaranteed to be broadcast. Thus, it is not suitable for applications where external hardware must be turned on when the system is powered on.

---

### **sysNotifyEarlyWakeupsEvent Specific Data**

None.

## Notifications

### Notification Reference

---

**Compatibility** Implemented only if [Notification Feature Set](#) is present.

## **sysNotifyEventDequeuedEvent**

The sysNotifyEventDequeuedEvent is broadcast for each event removed from the event queue with EvtGetEvent.

**WARNING!** Be very careful about registering for this notification; it can result in significantly degraded system performance.

---

### **sysNotifyEventDequeuedEvent Specific Data**

notifyDetailsP points to the dequeued event's EventType structure.

**IMPORTANT:** For speed, the event structure that notifyDetailsP points to uses system-native endianness. This means that you might need to byte-swap the structure's contents, depending on the endianness of the underlying operating system.

---

**Compatibility**

This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don't broadcast this notification. Palm OS 5 does broadcast it. Later versions may or may not broadcast this notification.

## **sysNotifyForgotPasswordEvent**

The sysNotifyForgotPasswordEvent is broadcast after the user taps the Lost Password button in the Security application. The notification is sent after the user has confirmed that all private records should be deleted but before the deletion actually occurs.

### **sysNotifyForgotPasswordEvent Specific Data**

None.

**Compatibility**

Implemented only if [Notification Feature Set](#) is present.

## **sysNotifyGotUsersAttention**

The `sysNotifyGotUsersAttention` notification is broadcast when the Attention Manager has finished displaying or sounding its attention indicators (blinking, playing sounds, vibrating, and so on).

System extensions or shared libraries should register for this notification if they want to perform some extra effect or if they simply want to be informed of when the user's attention was received.

### **sysNotifyGotUsersAttention Specific Data**

`notifyDetailsP` points to an `AttnNotifyDetailsType` structure.

<b>Prototype</b>	<pre>typedef struct {     AttnFlagsType flags; } AttnNotifyDetailsType;</pre>
<b>Fields</b>	<b>flags</b> The attention indicators that were used to get the user's attention. See <a href="#">AttnFlagsType</a> .
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.

## **sysNotifyHelperEvent**

The `sysNotifyHelperEvent` is broadcast by applications to request a service from another application. For example, the Address Book application broadcasts this notification to request that the Dial application dial a phone number. For the `sysNotifyHelperEvent`, the notification client (that is, the application or shared library that registers for the notification) is called a **helper**.

The application that broadcasts this notification specifies one of the action codes listed in [Table 35.1](#) in [Chapter 35, “Helper API”](#). These action codes request all helper applications to enumerate (list the services they perform), validate (ensure that the service will succeed), and execute (perform the action). The helper responds to the notification by returning the required data in the appropriate

## Notifications

### Notification Reference

---

portion of the `notifyDetailsP` structure and by setting the `handled` field to `true` or `false` to indicate the success or failure of the action.

For more information on this notification, see the section “[Helper Notifications](#)” on page 38 in the *Palm OS Programmer’s Companion*, vol. I.

#### **sysNotifyHelperEvent Specific Data**

`notifyDetailsP` points to a [HelperNotifyEventType](#) structure.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

### **sysNotifyIdleTimeEvent**

The `sysNotifyIdleTimeEvent` is broadcast when the system is idle and is about to doze.

**Compatibility** This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don’t broadcast this notification. Palm OS 5 does broadcast it. Later versions may or may not broadcast this notification.

### **sysNotifyInsPtEnableEvent**

The `sysNotifyInsPtEnableEvent` is broadcast at the start of `InsPtEnable`.

#### **sysNotifyInsPtEnableEvent Specific Data**

`notifyDetailsP` points to a Boolean: the `enableIt` parameter passed to `InsPtEnable`.

**Compatibility** This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don’t broadcast this notification. Palm OS 5 does broadcast it. Later versions may or may not broadcast this notification.

## **sysNotifyKeyboardDialogEvent**

The sysNotifyKeyboardDialogEvent is broadcast whenever the system keyboard is displayed. It is intended to enable the replacement of SysKeyboardDialog function's user interface.

### **sysNotifyKeyboardDialogEvent Specific Data**

notifyDetailsP points to the KeyboardType enum that indicates the mode in which the keyboard should be opened: alphabetic, numeric, or international.

#### **Compatibility**

This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don't broadcast this notification. Palm OS 5 does broadcast it. Later versions may or may not broadcast this notification.

## **sysNotifyLateWakeUpEvent**

The sysNotifyLateWakeUpEvent is broadcast during [SysHandleEvent](#) immediately after the device has finished waking up. This notification is sent at the late stage of wakeup, after the screen has been turned on. When this notification is broadcast, the system is guaranteed to fully wake up. Register for this notification if you need to perform startup tasks each time the system wakes up.

---

**IMPORTANT:** This notification is **not** guaranteed to be broadcast. Thus, it is unsuitable for applications where external hardware must be powered on when the device wakes up.

---

When the device receives this notification, it may be locked and waiting for the user to enter the password. If this is the case, you must wait for the user to unlock the device before you display a user interface. Therefore, if you intend to display a user interface when the device wakes up, you should make sure the device is not locked. If the device is locked, you should register for [sysNotifyDeviceUnlocked](#) notification and display your user interface when it is received. For example:

## Notifications

### Notification Reference

---

```
case sysNotifyLateWakeupEvent:
    if ((Boolean)
        PrefGetPreference(prefDeviceLocked) ) {
        SysNotifyRegister(myCardNo, myDbID,
                          sysNotifyDeviceUnlocked, NULL,
                          sysNotifyNormalPriority, NULL);
    } else {
        HandleDeviceWakeup();
    }
case sysNotifyDeviceUnlocked:
    HandleDeviceWakeup();
```

Note that the sysNotifyDeviceUnlocked notification is only broadcast on Palm OS 4.0 and higher.

#### **sysNotifyLateWakeupEvent Specific Data**

None.

<b>Compatibility</b>	Implemented only if <a href="#">Notification Feature Set</a> is present.
----------------------	--

#### **sysNotifyLocaleChangedEvent**

The sysNotifyLocaleChangedEvent is broadcast immediately after the system locale has changed. Currently, the user has the opportunity to change the locale only when the device first starts up and after a hard reset.

RAM-based applications and other code resources should obtain locale information by passing the prefLocale constant to [PrefGetPreference](#). They should not register for this notification. This notification is used by the built-in applications, which respond to it by rebuilding their default databases to use the newly selected language and character set.

#### **sysNotifyLocaleChangedEvent Specific Data**

notifyDetailsP points to a SysNotifyLocaleChangedType structure.

#### **Prototype**

```
typedef struct SysNotifyLocaleChangedTag {
    LmLocaleType oldLocale;
    LmLocaleType newLocale;
```

```
} SysNotifyLocaleChangedType;
```

<b>Fields</b>	oldLocale	The old locale. See <a href="#">LmLocaleType</a> .
	newLocale	The new locale.

<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
----------------------	---

## sysNotifyMenuBarOpenEvent

The sysNotifyMenuBarOpenEvent is broadcast during [MenuHandleEvent](#) when it is about to display the menu shortcut command bar.

Register for this notification if you are writing a system extension (such as a “hack” installed with the HackMaster program) that needs to add a button to the menu command bar or to suppress the menu command bar. To add a button, call [MenuCmdBarAddButton](#). To suppress the command toolbar, set the handled field to true.

Applications that need to add their own buttons to the menu command bar should do so in response to a [menuCmdBarOpenEvent](#). They should **not** register for this notification because an application should only add buttons if it is already the active application. The notification is sent after the event has been received, immediately before the command toolbar is displayed.

### sysNotifyMenuBarOpenEvent

None.

<b>Compatibility</b>	Implemented only if <a href="#">Notification Feature Set</a> is present.
----------------------	--

## sysNotifyNetLibIFMediaEvent

The sysNotifyNetLibIFMediaEvent is broadcast at the top of the event loop whenever the network interface makes the network connection active or inactive. The Network Panel uses this notification to decide whether the Connect button should be active.

## Notifications

### Notification Reference

---

Register for this notification if you need to know when the network connection is currently active.

#### **sysNotifyNetLibIFMediaEvent Specific Data**

notifyDetailsP contains a SysNotifyNetLibIFMediaType structure.

**Prototype**

```
typedef struct SysNotifyNetLibIFMediaTag {  
    NetLibIFMediaEventNotificationTypeEnum eType;  
    UInt32 ifCreator;  
    UInt16 ifInstance;  
} SysNotifyNetLibIFMediaType;
```

<b>Fields</b>	eType	One of the following values:  netIFMediaUp The network connection is active. This is usually sent after the network interface has displayed UI indicating that a connection attempt is in progress.  netIFMediaDown The network connection is inactive. This is usually sent after the network interface has brought the connection down because an inactivity timeout value was reached.  ifCreator ifInstance
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.	

#### **sysNotifyProcessPenStrokeEvent**

The sysNotifyProcessPenStrokeEvent is broadcast to enable custom recognition of strokes made on the silkscreen portion of the digitizer.

### **sysNotifyProcessPenStrokeEvent Specific Data**

notifyDetailsP points to a `SysNotifyPenStrokeType` structure.

#### **Prototype**

```
typedef struct SysNotifyPenStrokeTag {  
    UInt32      version;  
    PointType   startPt;  
    PointType   endPt;  
} SysNotifyPenStrokeType;
```

#### **Fields**

version	The current version of this structure. The current version is 0.
startPt	Start point of stroke.
endPt	End point of stroke.

#### **Compatibility**

This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don't broadcast this notification. Palm OS 5 does broadcast it. Later versions may or may not broadcast this notification.

### **sysNotifyResetFinishedEvent**

The `sysNotifyResetFinishedEvent` is broadcast immediately after the system has finished a reset.

Because the notification registry is cleared upon a reset, only internal system components use this notification. Applications that need to be informed of a system reset can respond to the [`sysAppLaunchCmdSystemReset`](#) launch code.

### **sysNotifyResetFinishedEvent Specific Data**

None.

#### **Compatibility**

Implemented only if [Notification Feature Set](#) is present.

### **sysNotifyRetryEnqueueKey**

The `sysNotifyRetryEnqueueKey` notification is broadcast at the top of the event loop if the Attention Manager has attempted to post

## Notifications

### Notification Reference

---

a virtual character to the key queue and failed because the queue is full. The notification signals that the Attention Manager is going to retry enqueueing the virtual character until it is successful.

Most applications do not need to register for this notification. It is used only by the Attention Manager to schedule retries of enqueueing the virtual character. When enqueueing a virtual character fails, the Attention Manager retries at the top of the event loop. It uses this notification to schedule retries so that they occur even if the user switches applications.

#### **sysNotifyRetryEnqueueKey Specific Data**

`notifyDetailsP` points to a `WChar` containing the virtual character to be enqueued.

<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
----------------------	---

#### **sysNotifySleepNotifyEvent**

The `sysNotifySleepNotifyEvent` is broadcast during [`SysHandleEvent`](#) immediately before the system is put to sleep. After the broadcast is complete, the system is put to sleep.

Register for this notification if you have a small amount of cleanup that needs to be performed before the system goes to sleep. It is recommended that you not perform any sort of prolonged activity, such as displaying an alert panel that requests confirmation, in response to a sleep notification. If you do, the alert might be displayed long enough to trigger another auto-off event, which could be detrimental to other handlers of this notification.

If your code is in the middle of a lengthy computation and needs to defer sleep, it should register for the [`sysNotifySleepRequestEvent`](#) instead.

---

**IMPORTANT:** This notification is **not** guaranteed to be broadcast. For example, if the system goes to sleep because the user removes the batteries, sleep notifications are not sent. Thus, these notifications are unsuitable for applications where external hardware must be shut off to conserve power before the system goes to sleep.

---

### **sysNotifySleepNotifyEvent Specific Data**

None.

**Compatibility** Implemented only if [Notification Feature Set](#) is present.

### **sysNotifySleepRequestEvent**

The `sysNotifySleepRequestEvent` is broadcast during [SysHandleEvent](#) processing when the system has decided to go to sleep.

Register for this notification if you need to delay the system from going to sleep while your code performs a lengthy operation, such as disconnecting from the network. The system checks the `deferSleep` value when each notification handler returns. If it is nonzero, it cancels the sleep event.

After you defer sleep, your code is free to finish what it was doing. When it is finished, you must allow the system to continue with the sleep event. To do so, create a [keyDownEvent](#) with the `resumeSleepChr` and the command key bit set (to signal that the character is virtual) and add it to the event queue. When the system receives this event, it will again broadcast the `sysNotifySleepRequestEvent` to all clients. If `deferSleep` is 0 after all clients return, then the system knows it is safe to go to sleep, and it broadcasts the `sysNotifySleepNotifyEvent` to all of its clients.

Note that you may receive this notification several times before the system goes to sleep because notification handlers can delay the system sleep and resume it later.

## Notifications

### Notification Reference

---

**IMPORTANT:** This notification is **not** guaranteed to be broadcast. For example, if the system goes to sleep because the user removes the batteries, sleep notifications are not sent. Thus, these notifications are unsuitable for applications where external hardware must be shut off to conserve power before the system goes to sleep.

---

#### **sysNotifySleepRequestEvent Specific Data**

notifyDetailsP points to a SleepEventParamType structure.

<b>Prototype</b>	<pre>typedef struct {     UInt16 reason;     UInt16 deferSleep; } SleepEventParamType;</pre>				
<b>Fields</b>	<table><tr><td><b>reason</b></td><td>The reason the system is going to sleep. The possible values are:  sysSleepAutoOff The idle time limit has been reached.  sysSleepPowerButton The user pressed the power off button.  sysSleepResumed The sleep event was deferred by one of the notification handlers but has been resumed through the use of the resumeSleepChr.  sysSleepUnknown Unknown reason.</td></tr><tr><td><b>deferSleep</b></td><td>Initially set to 0. If a notification handler wants to defer sleep, then it should increment this value. When deferSleep is greater than 0, the system waits before going to sleep.</td></tr></table>	<b>reason</b>	The reason the system is going to sleep. The possible values are:  sysSleepAutoOff The idle time limit has been reached.  sysSleepPowerButton The user pressed the power off button.  sysSleepResumed The sleep event was deferred by one of the notification handlers but has been resumed through the use of the resumeSleepChr.  sysSleepUnknown Unknown reason.	<b>deferSleep</b>	Initially set to 0. If a notification handler wants to defer sleep, then it should increment this value. When deferSleep is greater than 0, the system waits before going to sleep.
<b>reason</b>	The reason the system is going to sleep. The possible values are:  sysSleepAutoOff The idle time limit has been reached.  sysSleepPowerButton The user pressed the power off button.  sysSleepResumed The sleep event was deferred by one of the notification handlers but has been resumed through the use of the resumeSleepChr.  sysSleepUnknown Unknown reason.				
<b>deferSleep</b>	Initially set to 0. If a notification handler wants to defer sleep, then it should increment this value. When deferSleep is greater than 0, the system waits before going to sleep.				
<b>Compatibility</b>	Implemented only if <a href="#">Notification Feature Set</a> is present.				

## **sysNotifySyncFinishEvent**

The sysNotifySyncFinishEvent is broadcast immediately after a HotSync operation has completed. Register for this notification if you need to perform post-processing after HotSync operations.

### **sysNotifySyncFinishEvent Specific Data**

None.

<b>Compatibility</b>	Implemented only if <a href="#">Notification Feature Set</a> is present.
----------------------	--

## **sysNotifySyncStartEvent**

The sysNotifySyncStartEvent is broadcast immediately before a HotSync operation is begun. Register for this notification if you need to perform preprocessing before a HotSync operation.

### **sysNotifySyncStartEvent Specific Data**

None.

<b>Compatibility</b>	Implemented only if <a href="#">Notification Feature Set</a> is present.
----------------------	--

## **sysNotifyTimeChangeEvent**

The sysNotifyTimeChangeEvent notification is broadcast just after the system time has been changed using [TimSetSeconds](#). Register for this notification if you need to know when the time has changed.

### **sysNotifyTimeChangeEvent Specific Data**

None.

<b>Compatibility</b>	Implemented only if <a href="#">Notification Feature Set</a> is present.
----------------------	--

## **sysNotifyVirtualCharHandlingEvent**

The sysNotifyVirtualCharHandlingEvent is broadcast to enable custom handling of virtual characters.

## Notifications

### Notification Reference

---

#### **sysNotifyVirtualCharHandlingEvent Specific Data**

notifyDetailsP points to a `SysNotifyVirtualCharHandlingType` structure.

**Prototype**

```
typedef struct SysNotifyVirtualCharHandlingTag{  
    UInt32 version;  
    struct _KeyDownEventType keyDown;  
} SysNotifyVirtualCharHandlingType;
```

**Fields**

version	The current version of this structure. The current version is 0.
keyDown	The virtual character. See the description of the <code>keyDownEvent</code> in the Event Reference section of the <i>Palm OS Programmer's API Reference</i> for a complete description of this structure and its contents.

**Compatibility**

This notification is declared in the Palm OS 4.0 SDK Update 1. Versions 4.1 and earlier of Palm OS don't broadcast this notification. Palm OS 5 does broadcast it. Later versions may or may not broadcast this notification.

#### **sysNotifyVolumeMountedEvent**

The `sysNotifyVolumeMountedEvent` is broadcast when a Virtual File System Manager volume is mounted. When a volume is mounted, the VFS Manager activates the `start.prc` application on the newly mounted volume and switches applications to the Launcher or to the `start.prc` application on that volume if it has a user interface.

Register for this notification if you need to know when a volume is mounted or if you want to prevent the default behavior of the VFS Manager.

To prevent the VFS Manager from activating the `start.prc` application, set the `vfsHandledStartPrc` bit in the `handled` field. To prevent the VFS Manager from switching applications, set the `vfsHandledUIAppSwitch` bit.

### **sysNotifyVolumeMountedEvent Specific Data**

notifyDetailsP points to a [VFSSlotMountParamType](#) or [VFSPOSEMountParamType](#) structure.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

### **sysNotifyVolumeUnmountedEvent**

The sysNotifyVolumeUnmountedEvent is broadcast when a Virtual File System Manager volume is unmounted. Register for this notification if you need to know when a volume is unmounted.

#### **sysNotifyVolumeUnmountedEvent Specific Data**

notifyDetailsP points to a UInt16 containing the volume reference number.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## **Notifications**

### *Notification Reference*

---

# Attention Manager

---

This chapter provides reference material for the Attention Manager, and is divided into the following sections:

- [Attention Manager Data Structures](#)
- [Attention Manager Constants](#)
- [Attention Manager Functions](#)
- [Application-Defined Functions](#)

The Attention Manager API is declared in the header file `AttentionMgr.h`.

For more information about the attention manager, see the section “[Getting the User’s Attention](#)” in the *Palm OS Programmer’s Companion*, vol. I.

---

**IMPORTANT:** The Attention Manager was introduced in Palm OS® 4.0 and is not available in earlier versions of the operating system.

---

## Attention Manager Data Structures

### AttnCommandType

The `AttnCommandType` typedef specifies the set of possible commands that can be sent to the application requesting the user’s attention, either as a parameter to the [AttnCallbackProc](#) callback function or as one of the arguments that accompanies a [sysAppLaunchCmdAttention](#) launch code.

```
typedef UInt16 AttnCommandType;
```

The following table lists the values that `AttnCommandType` can assume.

## Attention Manager

### Attention Manager Data Structures

---

Constant	Description
kAttnCommandDrawDetail	Indicates that the application needs to draw the detailed contents of the attention dialog. The command arguments parameter points to a structure of type <a href="#">drawDetail</a> .
kAttnCommandDrawList	Indicates that the application needs to draw the appropriate list item in the attention dialog. The command arguments parameter points to a structure of type <a href="#">drawList</a> .
kAttnCommandPlaySound	Indicates that the application needs to play a sound. The command arguments parameter is NULL.
kAttnCommandCustomEffect	Indicates that the application needs to perform any application-specific special effects. This command is only sent for attention items that set the kAttnFlagsCustomEffectBit when they call <a href="#">AttnGetAttention</a> , which most applications won't do.
kAttnCommandGoThere	Tells the application to navigate to the item. The command arguments parameter is NULL. An application commonly calls <a href="#">SysAppLaunch</a> upon receipt of this command to have itself launched.
kAttnCommandGotIt	Tells the application that the user is dismissing the item. The command arguments parameter points to a structure of type <a href="#">gotIt</a> . The application may choose to clean up memory at this point.

Constant	Description
kAttnCommandSnooze	Indicates to the application that the user is snoozing. The command arguments parameter is NULL. Most applications do nothing upon receipt of this command. This command is passed to the appropriate application for each and every item currently pending, insistent or subtle. Applications with more than one attention item pending are called more than once.
kAttnCommandIterate	This command is passed to the application during the enumeration of attention items. It is particularly useful after HotSync® operations, as it allows the application to examine each item, updating or removing those that are stale or invalid. The command arguments parameter points to a structure of type <a href="#">iterate</a> .

## AttnCommandArgsType

The AttnCommandArgsType structure is a union of C structures. How you interpret the union's contents depends on which command it accompanies. Not all commands are accompanied by an AttnCommandArgsType structure, as listed in the following table:

AttnCommandType	Accompanied By
kAttnCommandDrawDetail	<a href="#">drawDetail</a>
kAttnCommandDrawList	<a href="#">drawList</a>
kAttnCommandPlaySound	None
kAttnCommandCustomEffect	None
kAttnCommandGoThere	None
kAttnCommandGotIt	<a href="#">gotIt</a>
kAttnCommandSnooze	None
kAttnCommandIterate	<a href="#">iterate</a>

## **Attention Manager**

### *Attention Manager Data Structures*

---

The structures that make up the `AttnCommandArgsType` union are described in the following sections.

#### **drawDetail**

When `kAttnCommandDrawDetail` is passed to the application, either via the callback function or as a parameter accompanying the `sysAppLaunchCmdAttention` launch code, the application needs to draw the detailed contents of the attention dialog. The `drawDetail` structure accompanies the `kAttnCommandDrawDetail` command, and provides the information needed to draw the contents of that dialog.

```
struct AttnCommandArgsDrawDetailTag {  
    RectangleType bounds;  
    Boolean firstTime;  
    AttnFlagsType flags;  
} drawDetail;
```

## Field Descriptions

bounds	Contains the window-relative bounding box for the area to draw. The clipping region is also set to the dimensions of this box to prevent accidental drawing outside.
firstTime	Set to <code>true</code> if the user has not yet seen this item. The value of this field could be used to display attentions that the user hasn't seen before in some special way. <code>firstTime</code> also indicates to your application whether or not the area in which your application is to draw has been erased. If <code>firstTime</code> is <code>false</code> , the area is not guaranteed to be blank; your application will need to erase it.
flags	The global user preferences for this attention attempt combined with the custom flags passed in by the developer. Only the lower 16 bits of this field have meaning; use <code>kAttnFlagsSoundBit</code> , <code>kAttnFlagsLEDBit</code> , <code>kCustomFlagsVibrateBit</code> , and <code>kAttnFlagsCustomEffectBit</code> (described under “ <a href="#">AttnFlagsType</a> ” on page 111) to interpret the contents of this field.

## drawList

When `kAttnCommandDrawList` is passed to the application, either via the callback function or as a parameter accompanying the [`sysAppLaunchCmdAttention`](#) launch code, the application needs to draw the appropriate list item in the attention dialog. The `drawList` structure accompanies the `kAttnCommandDrawList` command, and provides the information needed to draw the contents of that dialog.

```
struct AttnCommandArgsDrawListTag {  
    RectangleType bounds;  
    Boolean firstTime;  
    AttnFlagsType flags;
```

## Attention Manager

### Attention Manager Data Structures

---

```
    Boolean selected;
} drawList;
```

#### Field Descriptions

bounds	Contains the window-relative bounding box for the area to draw. The clipping region is also set to the dimensions of this box to prevent accidental drawing outside.
firstTime	Set to true if the user has not yet seen this item. The value of this field could be used, for example, to trigger a custom sound the first time this attention item is presented to the user.
flags	The global user preferences for this attention attempt combined with the custom flags passed in by the developer. Only the lower 16 bits of this field have meaning; use kAttnFlagsSoundBit, kAttnFlagsLEDBit, kCustomFlagsVibrateBit, and kAttnFlagsCustomEffectBit (described under “ <a href="#">AttnFlagsType</a> ” on page 111) to interpret the contents of this field.
selected	Set to true if the item has been selected. It is up to your code to draw the item appropriately (typically by changing the UI colors) based upon the value of this flag.

#### gotIt

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

```
struct AttnCommandArgsGotItTag {  
    Boolean dismissedByUser;  
} gotIt;
```

### Field Descriptions

dismissedByUser	true indicates that the user dismissed the attention. false indicates that the kAttnCommandGotIt command was generated by a call to <a href="#">AttnForgetIt</a> .
-----------------	--

### iterate

When kAttnCommandIterate is passed to the application, either via the callback function or as a parameter accompanying the [sysAppLaunchCmdAttention](#) launch code, it is accompanied by an iterate structure. This structure contains any necessary data that the application may need in order to process the callback or launch code.

```
struct AttnCommandArgsIterateTag {  
    UInt32 iterationData;  
} iterate;
```

### Field Descriptions

iterationData	Any necessary data that the application may need in order to process the callback or launch code. The value of this field is that which was originally passed to <a href="#">AttnIterate</a> .
---------------	--

## AttnFlagsType

Pass a value of this type to [AttnGetAttention](#) and [AttnUpdate](#) to specify what means the device should or should not use to get the user's attention. A value of this type is also passed back to your code either as a parameter to the callback function or, if no callback was specified, as part of the structure passed with the [sysAppLaunchCmdAttention](#) launch code.

## Attention Manager

### Attention Manager Data Structures

---

```
typedef UInt32 AttnFlagsType;
```

Note that more bits may be defined if necessary to accommodate additional hardware.

Constant	Value	Description
kAttnFlagsSoundBit	0x0001	Plays a sound.
kAttnFlagsLEDBit	0x0002	Blinks an LED, if the device is so equipped.
kAttnFlagsVibrateBit	0x0004	Triggers vibration, if the device is so equipped.
kAttnFlagsCustomEffectBit	0x0008	Triggers an application-specific custom effect.
kAttnFlagsAllBits	0xFFFF	Uses all available means to get the user's attention.
kAttnFlagsUseUserSettings	0x0000	System-wide preferences determine what means are used to get the user's attention.

The following constant values can be used to override the user's settings and force or prevent specific behaviors:

Constant	Value	Description
kAttnFlagsAlwaysSound	kAttnFlagsSoundBit	Play a sound, regardless of the user's settings.
kAttnFlagsAlwaysLED	kAttnFlagsLEDBit	Blink an LED, if the device is so equipped, regardless of the user's settings.
kAttnFlagsAlwaysVibrate	kAttnFlagsVibrateBit	Vibrate, if the device is so equipped, regardless of the user's settings.
kAttnFlagsAlwaysCustomEffect	kAttnFlagsCustomEffectBit	Trigger an application-specific custom effect.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
kAttnFlagsEverything	kAttnFlagsAllBits	Use every available means to get the user's attention, regardless of the user's settings.
kAttnFlagsNoSound	kAttnFlagsSoundBit $\ll 16$	Prevent a sound from being played, regardless of the user's settings.
kAttnFlagsNoLED	kAttnFlagsLEDBit $\ll 16$	Prevent the LED from flashing, regardless of the user's settings.
kAttnFlagsNoVibrate	kAttnFlagsVibrateBit $\ll 16$	Prevent vibration, regardless of the user's settings.
kAttnFlagsNoCustomEffect	kAttnFlagsCustomEffectBit $\ll 16$	Prevent triggering of the application-specific custom effect.
kAttnFlagsNothing	kAttnFlagsAllBits $\ll 16$	Disable all attention-getting mechanisms, regardless of the user's settings.

These constants can be used in combination. For example, to disable both sound and the LED, use  
`kAttnFlagsNoSound | kAttnFlagsNoLED`.

If neither `kAttnFlagsAlwaysSound` nor `kAttnFlagsNoSound` is set for a given attention item, a sound plays if and only if the user's preference is to play a sound and the user's preference for alarm volume is non-zero.

## AttnLaunchCodeArgsType

If a callback function is not specified in a call to [AttnGetAttention](#) and the Attention Manager needs your code to draw the details of your attention in the attention dialog or perform another attention-specific function, it sends a

## Attention Manager

### Attention Manager Data Structures

---

[sysAppLaunchCmdAttention](#) launch code to your application.

Along with the launch code, it passes a pointer to the following structure, which indicates both what your code is expected to do and identifies the attention that triggered the launch code:

```
typedef struct {
    AttnCommandType command;
    UInt32 userData;
    AttnCommandArgsType *commandArgsP;
} AttnLaunchCodeArgsType;
```

#### Field Descriptions

command	Indicates what your code is being requested to do. The complete list of possible commands are described in the definition of <a href="#"><u>AttnCommandType</u></a> .
userData	Identifier that distinguishes the particular attention item from others made by this application. This identifier was specified when the attention item was created.
commandArgsP	Pointer to command-specific arguments. See the description of each command for a discussion of that command's arguments.

When processing the launch code be aware that your application doesn't have application globals available to it; it is important that anything necessary to draw or otherwise display be available through commandArgsP.

## AttnLevelType

Attention attempts can either be insistent or subtle. Insistent attention attempts make a serious effort to get the user's attention, by both displaying a dialog and possibly by triggering one or more special effects, such as blinking a light, vibrating, or playing a sound. Other alerts are of a less serious nature and shouldn't disrupt the user. Consequently, subtle attention attempts typically make the attention indicator blink and may trigger one or more special effects, but don't display the Attention Manager dialog. The user can then work until a suitable time, at which point they can tap

on the indicator to see what needs their attention. Subtle attention attempts might be used for telling the user that they have new e-mail, or perhaps that a holiday or birthday is coming up.

```
typedef UInt16 AttnLevelType;
```

The following table lists the two defined values for AttnLevelType:

Constant	Description
kAttnLevelInsistent	An insistent attention attempt. Make a serious effort to get the user's attention by displaying a dialog and optionally triggering one or more special effects.
kAttnLevelSubtle	A subtle attention attempt. Notify the user using special effects, but don't disrupt the user with the dialog if the device is in use.

Note that user preferences for the various special effects can't be set separately for subtle and insistent attention attempts. If your application needs to vary the effects based upon the AttnLevelType, pass a suitable value for the flags parameter in your [AttnGetAttention](#) call.

## Attention Manager Constants

In addition to the constant values defined specifically for use with [AttnCommandType](#), [AttnFlagsType](#), and [AttnLevelType](#), the Attention Manager defines the following constant types:

- [Error Code Constants](#)
- [Attention Manager Drawing Constants](#)
- [Attention Manager Feature Constants](#)

### Error Code Constants

The Attention Manager returns the following error code under the circumstances described below.

## Attention Manager

### Attention Manager Constants

---

Constant	Description
attnErrMemory	Returned by <a href="#">AttnGetAttention</a> when there is insufficient memory to perform the requested operation.

## Attention Manager Drawing Constants

The following four constants define the on-screen boundaries of the attention indicator:

Constant	Value	Description
kAttnIndicatorLeft	0	The left-hand edge of the attention indicator.
kAttnIndicatorTop	0	The top-most edge of the attention indicator.
kAttnIndicatorWidth	16	The width of the attention indicator.
kAttnIndicatorHeight	16	The height of the attention indicator.

The following two constants are used when drawing the list view. Applications should use these constants to format the display of information in the Attention Manager's list view. Draw the application's small icon centered within the first kAttnListMaxIconWidth pixels of the drawing bounds. Then draw two lines of text describing the attention, left-justified, starting at kAttnListTextOffset from the left edge of the drawing bounds.

Constant	Value	Description
kAttnListMaxIconWidth	15	Maximum width of the application's icon. If the icon is narrower than this, it should be drawn centered within this width.
kAttnListTextOffset	17	Offset, from the left-hand edge of the drawing bounds, of the textual description of the attention.

## Attention Manager Feature Constants

The Attention Manager defines a read-only feature ('attn', 0) that indicates the current user settings and capabilities of the hardware. The upper 16 bits of the feature indicate whether or not the hardware has the capability to perform that sort of alert. The lower 16 bits indicate whether the user has currently enabled that sort of alert.

Constant	Value	Description
kAttnFtrCreator	'attn'	Attention Manager feature creator.
kAttnFtrCapabilities	0	Attention Manager feature number.

When working with the value obtained with `FtrGet`, use the following two constants to separate those bits that contain the user settings from those bits that identify the device's capabilities:

Constant	Value	Description
kAttnFlagsUserSettingsMask	<code>kAttnFlagsAllBits</code>	Mask to isolate those bits that contain the user settings.
kAttnFlagsCapabilitiesMask	<code>kAttnFlagsAllBits &lt;&lt; 16</code>	Mask to isolate those bits that contain the device capabilities.

These constants can be used to interpret the device capabilities (`kAttnFlagsHas...`) and the user settings (`kAttnFlagsUserWants...`):

## Attention Manager

### Attention Manager Constants

---

Constant	Value	Description
kAttnFlagsHasLED	kAttnFlagsLEDBit << 16	The device has an LED that can be illuminated to indicate an alert.
kAttnFlagsHasSound	kAttnFlagsSoundBit << 16	The device is capable of playing a sound to indicate an alert.
kAttnFlagsHasVibrate	kAttnFlagsVibrateBit << 16	The device is capable of vibrating to indicate an alert.
kAttnFlagsHasCustomEffect	kAttnFlagsCustomEffectBit << 16	Not used.
kAttnFlagsUserWantsLED	kAttnFlagsLEDBit	The user wants the LED illuminated to signal an alert.
kAttnFlagsUserWantsSound	kAttnFlagsSoundBit	The user wants a sound played to signal an alert.
kAttnFlagsUserWantsVibrate	kAttnFlagsVibrateBit	The user wants the device to vibrate to signal an alert.
kAttnFlagsUserWantsCustomEffect	kAttnFlagsCustomEffectBit	Not used.

# Attention Manager Functions

## AttnDoSpecialEffects

<b>Purpose</b>	Triggers an Attention Manager special effect set.
<b>Declared In</b>	AttentionMgr.h
<b>Prototype</b>	Err AttnDoSpecialEffects (AttnFlagsType flags)
<b>Parameters</b>	-> flags      Specifies the behavior to be exhibited by this special effects request. See <a href="#">AttnFlagsType</a> for the various bits that make up this flag. Note that the behavior is undefined if you set incompatible flags. Supply kAttnFlagsUseUserSettings to have this attention request follow the user's pre-set preferences.
<b>Result</b>	Returns errNone if no problems were encountered. Returns attnErrMemory if there wasn't enough memory to accommodate the attention request.
<b>Comments</b>	This routine is provided as a convenience for applications that need to trigger special effects. It does the equivalent of one "nag" of an Attention Manager special effect set.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.

## Attention Manager

### Attention Manager Functions

---

## AttnForgetIt

<b>Purpose</b>	Provides a way for applications to tell the Attention Manager to forget about an attention item.						
<b>Declared In</b>	AttentionMgr.h						
<b>Prototype</b>	Boolean AttnForgetIt (UInt16 cardNo, LocalID dbID, UInt32 userData)						
<b>Parameters</b>	<table><tr><td>-&gt; cardNo</td><td>Card number on which the application making the request resides.</td></tr><tr><td>-&gt; dbID</td><td>Database ID of the application making the request.</td></tr><tr><td>-&gt; userData</td><td>Identifier that distinguishes the attention attempt from others made by the same application. This identifier can be an integer, a pointer, or any other 32-bit value.</td></tr></table>	-> cardNo	Card number on which the application making the request resides.	-> dbID	Database ID of the application making the request.	-> userData	Identifier that distinguishes the attention attempt from others made by the same application. This identifier can be an integer, a pointer, or any other 32-bit value.
-> cardNo	Card number on which the application making the request resides.						
-> dbID	Database ID of the application making the request.						
-> userData	Identifier that distinguishes the attention attempt from others made by the same application. This identifier can be an integer, a pointer, or any other 32-bit value.						
<b>Result</b>	Returns <code>true</code> if the item was removed, <code>false</code> if a matching item was not found.						
<b>Comments</b>	<p>You typically call this function after your application has handled a “Go There” event and the user has read about the item. For example, if there is a subtle attention pending that says “you have three e-mail messages waiting” and you go to the e-mail application on your own and read your e-mail, the subtle notification must disappear. <code>AttnForgetIt</code> allows the application to do this.</p> <p>Note that this call can be made when the Attention Manager dialog is on-screen (though presumably that is rare, since the application is probably not doing much at this point). If this call removes a list item, then the Attention Manager may call back into other items to redraw the list.</p> <p>If this call removes the last item when any indicator is present, the indicator disappears. If this call removes the last unread item, but read items remain, the indicator switches from blinking to steady state.</p>						

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## AttnGetAttention

**Purpose** Requests the user's attention.

**Declared In** AttentionMgr.h

**Prototype**

```
Err AttnGetAttention (UInt16 cardNo,
LocalID dbID, UInt32 userData,
AttnCallbackProc *callbackFnP,
AttnLevelType level, AttnFlagsType flags,
UInt16 nagRateInSeconds, UInt16 nagRepeatLimit)
```

**Parameters**

-> cardNo	Card number on which the application making the request resides.
-> dbID	Database ID of the application making the request.
-> userData	Application-specific data that is later passed back to your code through the callback function. If no callback function is specified in the callbackFnP parameter, this data is included in what is passed along with a <a href="#">sysAppLaunchCmdAttention</a> launch code. userData can be an integer, a pointer, or any other 32-bit value as needed by your application. Most applications pass the unique ID or other key for the record which caused the attention request. userData is also used to distinguish a given attention attempt from others made by the same application.

## Attention Manager

### Attention Manager Functions

---

-> callbackFnP

Pointer to the function registered by the application to be called by the Attention Manager when the attention is displayed or removed. See [AttnCallbackProc](#), below, for the callback function's parameters. Supply NULL to instead have a [sysAppLaunchCmdAttention](#) launch code sent to the application that made the attention request whenever the attention is displayed or removed.

-> level

Indicates the annoyance level. Pass one of the values defined for [AttnLevelType](#).

-> flags

Behavior override, if necessary, for this attention request. This override allows, for instance, silent alarms or noisy alarms. See [AttnFlagsType](#) for the various bits that make up this flag. Note that the behavior is undefined if you set incompatible flags. Supply kAttnFlagsUseUserSettings to have this attention request follow the user's pre-set preferences.

-> nagRateInSeconds

How long to wait before nagging.

-> nagRepeatLimit

How many times to nag, excluding the first attempt.

**Result** Returns errNone if no problems were encountered. Returns attnErrMemory if there wasn't enough memory to accommodate the attention request.

**Comments** The combination of cardNo, dbID and userData uniquely identify an attention-getting attempt. If another call is made to AttnGetAttention with identical values for these arguments, an error is reported. To update or delete an existing attention item, pass these same values to [AttnUpdate](#) or [AttnForgetIt](#), respectively.

In response to `AttnGetAttention`, the behavior of the operating system or application depends on whether there already are other demands and on the annoyance level passed in the `AttnGetAttention` call.

- No other demands, insistent attention request:

The Attention Manager puts up a dialog that details the current attempt to get the user's attention.

- Other demands exist, insistent attention request:

The Attention Manager adds a summary of the current attempt to get the user's attention to a list of things that need attention. If the dialog is currently in detail form—which is the case if just one other demand exists—the view is refreshed, changing from detail to list form. In this case, the pen and key event queues are also flushed so that any user events that are happening while the display is changing are explicitly ignored. Two exceptions to this behavior exist: if all existing attentions are subtle, or if all existing insistent attentions were snoozed, the new insistent attention brings up the dialog in detail mode, rather than list mode.

- Subtle attention request:

The Attention Manager starts the attention indicator blinking, and adds the item to its list for later display, unless the dialog is currently being displayed in list mode. In this event, the new subtle attention item simply appears in the list; the indicator does not blink to announce it.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [AttnUpdate](#)

## Attention Manager

### Attention Manager Functions

---

## AttnGetCounts

**Purpose** Returns the number of attention items that are currently pending.

**Declared In** AttentionMgr.h

**Prototype** `UInt16 AttnGetCounts (UInt16 cardNo,  
LocalID dbID, UInt16 *insistentCountP, UInt16  
*subtleCountP)`

**Parameters**

-> cardNo	If this value is zero, counts pending attention items from applications on all cards. Otherwise, counts only pending attention items from applications on the specified card.
-> dbID	If this value is zero, counts pending attention items from all applications. Otherwise, counts only pending attention items from applications with the specified database ID.

`<- insistentCountP`  
Pointer to a 16-bit unsigned value that is filled in with the number of insistent items pending. Pass NULL for this parameter if you don't need to know the number of insistent items that are pending.

`<- subtleCountP`  
Pointer to a 16-bit unsigned value that is filled in with the number of subtle items pending. Pass NULL for this parameter if you don't need to know the number of subtle items that are pending.

**Result** Returns the total number of items, both insistent and subtle, that are currently pending.

**Comments** Call this function if you need to exhibit different behavior if attention items are already pending.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## **AttnIndicatorEnable**

**Purpose** Enables and disables the on-screen attention indicator.

**Declared In** AttentionMgr.h

**Prototype** void AttnIndicatorEnable (Boolean enableIt)

**Parameters** -> enableIt true to enable the attention indicator, false to disable it.

**Result** Returns nothing.

**Comments** This function is used by applications to enable or disable the on-screen attention indicator. The indicator only blinks when all of the following are true:

- The indicator is enabled.
- The indicator is being asked to blink by the attention manager.
- The operating system isn't using the display in such a way as to prevent the attention indicator from showing, such as when the menu bar is being displayed or when a modal dialog is on top of the form.

The attention indicator is enabled by default. If your application controls the upper portion of the screen and needs to prevent the attention indicator from being displayed, call [AttnIndicatorEnable](#) and pass it a value of `false`.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [AttnIndicatorEnabled](#)

## Attention Manager

### Attention Manager Functions

---

## AttnIndicatorEnabled

<b>Purpose</b>	Returns whether the on-screen attention indicator is currently enabled.
<b>Declared In</b>	AttentionMgr.h
<b>Prototype</b>	Boolean AttnIndicatorEnabled (void)
<b>Parameters</b>	None.
<b>Result</b>	Returns true if the on-screen attention indicator is currently enabled, false otherwise.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">AttnIndicatorEnable</a>

## AttnIterate

<b>Purpose</b>	Instructs the Attention Manager to check each attention item currently pending and, for those that match the specified card number and database ID, invoke the item's callback routine. If a callback routine was not specified in the request, the <a href="#">sysAppLaunchCmdAttention</a> launch code is sent to the application that made the attention request.				
<b>Declared In</b>	AttentionMgr.h				
<b>Prototype</b>	void AttnIterate (UInt16 cardNo, LocalID dbID, UInt32 iterationData)				
<b>Parameters</b>	<table><tr><td>-&gt; cardNo</td><td>Card number on which the application that made the request resides.</td></tr><tr><td>-&gt; dbID</td><td>Database ID of the application that made the request.</td></tr></table>	-> cardNo	Card number on which the application that made the request resides.	-> dbID	Database ID of the application that made the request.
-> cardNo	Card number on which the application that made the request resides.				
-> dbID	Database ID of the application that made the request.				

-> iterationData

Any necessary data that the application may need in order to process the callback or launch code. See the description of the [AttnCallbackProc](#) function for more information on this parameter.

**Result** Returns nothing.

**Comments** This function iterates through all of the attention requests made by this application and uses the callback or launch code for each to inform the application about the attention request. When an application receives a [sysAppLaunchCmdSyncNotify](#) launch code, signifying that a HotSync occurred that affected that application's databases, the application typically calls `AttnIterate` so it can remove attention requests for records that may have been removed during the HotSync. Applications can also call [AttnGetAttention](#) after a HotSync, if necessary.  
Note that you can call [AttnForgetIt](#) inside the iteration since it only marks the record for deletion and thus doesn't confuse the iteration.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## AttnListOpen

**Purpose** Displays the attention dialog in list mode and, after the user has dismissed it, acts accordingly based on how it was dismissed.

**Declared In** AttentionMgr.h

**Prototype** void AttnListOpen (void)

**Parameters** None.

**Result** Returns nothing.

## Attention Manager

### Attention Manager Functions

---

- Comments** This function allows applications that do not provide the blinking attention indicator to open the list, if necessary.
- Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## AttnUpdate

- Purpose** Updates one or more aspects of a specified attention item.
- Declared In** AttentionMgr.h
- Prototype** Boolean AttnUpdate (UInt16 cardNo, LocalID dbID,  
UInt32 userData, AttnCallbackProc \*callbackFnP,  
AttnFlagsType \*flagsP, UInt16 \*nagRateInSecondsP,  
UInt16 \*nagRepeatLimitP)
- Parameters**
- |             |  |
|-------------|--|
| -> cardNo   | Card number on which the application that made the request resides.  |
| -> dbID     | Database ID of the application that made the request.  |
| -> userData | Application-specific data that is passed back to your code through the callback function. If no callback function is specified in the callbackFnP parameter, this data is included in what is passed along with a <a href="#">sysAppLaunchCmdAttention</a> launch code. userData can be an integer, a pointer, or any other 32-bit value. Most applications pass the unique ID or other key for the record which caused the attention request. The value of the userData parameter is also used to distinguish a given attention attempt from others made by the same application. |

-> callbackFnP

Registers a new function to be called by the Attention Manager when the attention is displayed or removed. The function to which this parameter points should conform to [AttnCallbackProc](#). Supply NULL to instead have a [sysAppLaunchCmdAttention](#) launch code sent to the application that made the attention request whenever the attention is displayed or removed.

---

**IMPORTANT:** Because NULL indicates that a launch code should be sent whenever the callback would otherwise be invoked, it isn't used in this instance to leave the original setting for the `callbackFnP` parameter intact. The value supplied for this parameter *always* overwrites the value supplied in the original attention request.

---

-> flagsP

Pointer to a set of flags that can be used to override user-specified attention behavior; for instance, to force silent or noisy alarms. See [AttnFlagsType](#) for the various bits that make up this flag, and note that the behavior is undefined if you set incompatible flags. Pass NULL to leave the current flag settings unchanged.

-> nagRateInSecondsP

Pointer to the length of time to wait before nagging. Pass NULL to leave the "nag rate" unchanged.

-> nagRepeatLimitP

Pointer to the maximum number of times the user should be nagged. Pass NULL to leave the nag repeat limit unchanged.

**Result** Returns `true` if the update was successful, `false` if no matching attention item was found.

## Attention Manager

### *Application-Defined Functions*

---

<b>Comments</b>	This call may result in the callback function being called to re-display the item. If no callback function is specified, the <a href="#">sysAppLaunchCmdAttention</a> launch code is instead sent to your application. It may also result in callbacks to other pending attention requests.  You call <code>AttnUpdate</code> to tell the Attention Manager to update, forcing it to call into all of its clients to redraw. This provides a way for an application to update the text of an attention item without tearing down and re-opening the Attention Manager dialog. For example, <code>AttnUpdate</code> could be used to update an existing email notification to say "You have three new email messages" when additional messages are received.  Although <code>AttnUpdate</code> may cause a given attention item to redraw, it does not rerun the special effects (if any) that occurred when that attention item was added. If you want to trigger Attention Manager effects for a particular item, call <a href="#">AttnForgetIt</a> followed by <a href="#">AttnGetAttention</a> .
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">AttnGetAttention</a>

## Application-Defined Functions

### **AttnCallbackProc**

<b>Purpose</b>	Provides a function prototype to be used by callback functions supplied to <a href="#">AttnGetAttention</a> and <a href="#">AttnUpdate</a> . The supplied
----------------	---

function is invoked by the Attention Manager whenever the attention is displayed or removed.

**Declared In** AttentionMgr.h

**Prototype**

```
typedef Err AttnCallbackProc  
(AttnCommandType command, UInt32 userData,  
AttnCommandArgsType *commandArgsP)
```

<b>Parameters</b>	-> command	Indicates what the callback function is being requested to do. The complete list of possible commands are described in the definition of <a href="#">AttnCommandType</a> .
	-> userData	Identifier that distinguishes the particular attention item from others made by this application. This identifier was specified when the attention item was created.
	-> commandArgsP	Pointer to command-specific arguments. See the description of each command for a discussion of that command's arguments.

**Result** The callback function should return `errNone` if it correctly handled the command, or an appropriate error code otherwise. If the callback function returns an error code other than `errNone`, the attention is removed from the list of active attention items.

**Comments** For a given attention item, the Attention Manager calls back to the code resource that created that item whenever the Attention Manager needs the resource to draw the attention dialog contents or whenever it needs to inform the code resource of activity relating to the attention item. The Attention Manager calls back using one of two mechanisms:

- If a callback routine has been specified for a given attention item, the Attention Manager invokes the specified routine. This callback routine doesn't have application globals available to it, so it is important that anything necessary to draw or otherwise display be available through

## Attention Manager

### *Application-Defined Functions*

---

commandArgsP. A callback routine is typically used by libraries and system extensions.

- If a callback routine has not been specified for a given attention item, the Attention Manager instead sends a [sysAppLaunchCmdAttention](#) launch code to the application that registered the attention item. Accompanying that launch code is an [AttnLaunchCodeArgsType](#) structure containing the three parameters documented above. Applications typically use the launch-code mechanism due to the restrictions that are placed on callback routines.

---

**IMPORTANT:** It is your responsibility to ensure that the callback procedure is still in the same place when it gets called, dealing with the possibility that the code resource might be unlocked and moved in memory, and with the possibility that the database containing the code resource might be deleted. For the most part, these problems don't exist when using launch codes.

---

**Compatibility** Invoked only if [4.0 New Feature Set](#) is present.

# Categories

---

This chapter describes the category API as declared in the header file `Category.h`. It discusses the following topics:

- [Category Data Structures](#)
- [Category Constants](#)
- [Category Functions](#)

For more information on categories, see the section “[Categories](#)” on page 116 in the *Palm OS Programmer’s Companion*, vol. I.

## Category Data Structures

### AppInfoPtr

The `AppInfoPtr` defines a pointer to an [AppInfoType](#) structure.

```
typedef AppInfoType *AppInfoPtr;
```

### AppInfoType

The `AppInfoType` structure shown below maps category names to category indexes and unique IDs. To use the category API described in this chapter, a database’s application info block must either be an `AppInfoType` structure, or it must have an `AppInfoType` structure as its first field.

```
typedef struct {
    UInt16    renamedCategories;
    Char      categoryLabels [dmRecNumCategories]
              [dmCategoryLength];
    UInt8    categoryUniqIDs [dmRecNumCategories];
    UInt8    lastUniqID;
    UInt8    padding;
} AppInfoType;
```

## Categories

### *Category Constants*

---

Allocate the application info block in the storage heap and use the [DmSetDatabaseInfo](#) function to set the database's application info ID to the local ID of this structure. Then, use the [CategoryInitialize](#) function to initialize it with a localized list of strings containing the category names.

#### Field Descriptions

renamedCategories	Used by <a href="#">CategorySetName</a> as a bit field indicating which categories have been renamed. Usually cleared by a conduit.
categoryLabels	An array of strings containing the category names. The maximum size of the array is <code>dmRecNumCategories</code> , and the maximum length of each string in the array is <code>dmCategoryLength</code> . Both of these constants are defined in <code>DataMgr.h</code> .
categoryUniqIDs	Category IDs used for synchronization with the desktop database. Unique IDs generated by the device are between 0 and 127. Unique IDs generated by the desktop computer are between 128 and 255.
lastUniqID	Used for sorting and assigning unique IDs.

## Category Constants

The following category constants are defined:

<b>Constant</b>	<b>Value</b>	<b>Description</b>
categoryHideEditCategory	10000	Used to suppress the “Edit Categories” item.
categoryDefaultEditCategoryString	10001	Used to show the default “Edit Categories” item.

---

**NOTE:** These constants look like system resource IDs, but they are not. To use a non-default string for the “Edit Categories” item you pass a resource ID of a string containing your title. If you want to use the default or hide the item, you pass one of these constants. They are within the system resource ID range (that is, they are greater than 10000) so that they don’t conflict with any other possible value for that parameter.

---

**Compatibility** Both categoryHideEditCategory and categoryDefaultEditCategoryString are defined only if the [3.5 New Feature Set](#) is present.

## Categories

### Category Functions

---

## Category Functions

### CategoryCreateList

**Purpose** Populate a popup list with a database's categories.

**Declared In** Category.h

**Prototype**

```
void CategoryCreateList (DmOpenRef db,
ListType *listP, UInt16 currentCategory,
Boolean showAll, Boolean showUneditables,
UInt8 numUneditableCategories,
UInt32 editingStrID, Boolean resizeList)
```

<b>Parameters</b>	-> db	Open database containing the category information you want to read.
	<- listP	Pointer to the <a href="#">ListType</a> structure that should display the categories.
	-> currentCategory	Index of the category to select. The index is the index into the <code>categoryLabels</code> array. The default is to have the “Unfiled” category selected.
	-> showAll	true to include an “All” list item.
	-> showUneditables	true to show uneditable categories.
	-> numUneditableCategories	The number of categories that the user cannot edit. You should store uneditable categories at the beginning of the <code>categoryLabels</code> array. For example, it's common to have an “Unfiled” category at position zero that is not editable. This function displays the uneditable categories at the end of the popup list.

-> editingStrID The resource ID of a tSTR resource to use as the Edit Categories list item. To use the default string ("Edit Categories") pass the constant `categoryDefaultEditCategoryString`.

If you don't want users to edit categories, pass the `categoryHideEditCategory` constant.

-> resizeList true to resize the list to the number of categories. Set to true for popups, false otherwise.

**Result** Returns nothing.

**Comments** The "All" item is first in the list (if the `showAll` parameter is true), followed by the editable categories in the database and then the categories that cannot be edited. The option to edit categories is last in the list and can be suppressed if desired.

You rarely call this function directly. Instead, most applications use [CategorySelect](#), which calls this function and fully manages the user's selection of a category in the popup list. Use `CategoryCreateList` only if you want more control over the category popup list.

This function obtains the `db` parameter's `appInfoID`, reads the [AppInfoType](#) structure at that location, and uses the information in it to initialize the `listP`'s `items` array with the names of the database's categories. You must have already allocated the structure pointed to by `listP`. `CategoryCreateList` does not display the list; use [LstPopupList](#) or [LstDrawList](#) to do so.

You must balance a call to `CategoryCreateList` with a call to [CategoryFreeList](#). The `CategoryCreateList` function locks the resources for the category names. It also allocates the `listP` items array. `CategoryFreeList` unlocks all resources locked by `CategoryCreateList` and frees all memory allocated by `CategoryCreateList`.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

The constants `categoryDefaultEditCategoryString` and `categoryHideEditCategory` are defined only if [3.5 New](#)

## Categories

### Category Functions

---

[Feature Set](#) is present. In earlier versions, you can suppress the Edit Categories string by passing 0 for the editingStrID parameter, or include the item by passing categoryEditStrID.

**See Also** [CategoryCreateListV10](#)

## CategoryCreateListV10

**Purpose** Read a database's categories and set the category list.

**Declared In** Category.h

**Prototype** void CategoryCreateListV10 (DmOpenRef db,  
ListType \*lst, UInt16 currentCategory,  
Boolean showAll)

**Parameters**

-> db	Open database containing the category information you want to read.
<- lst	Pointer to the <a href="#">ListType</a> that should display the categories.
-> currentCategory	Index of the category to select. The index is the index into the categoryLabels array. The default is to have the “Unfiled” category selected.
-> showAll	true to include an “All” list item.

**Result** Returns nothing.

**Compatibility** This function corresponds to the Palm OS® 1.0 version of [CategoryCreateList](#). It is obsolete.

## CategoryEdit

<b>Purpose</b>	Event handler for the Edit Categories dialog.	
<b>Declared In</b>	Category.h	
<b>Prototype</b>	<pre>Boolean CategoryEdit (DmOpenRef db,                       UInt16 *category, UInt32 titleStrID,                       UInt8 numUneditableCategories)</pre>	
<b>Parameters</b>	<code>-&gt; db</code>	Open database containing the categories to be edited.
	<code>&lt;- category</code>	Upon return, the index of the last category selected before the dialog was closed.
	<code>-&gt; titleStrID</code>	The resource ID of a tSTR resource to use as the dialog's title. To use the default string ("Edit Categories"), pass the constant <code>categoryDefaultEditCategoryString</code> .
	<code>-&gt; numUneditableCategories</code>	The number of categories that the user cannot edit. You should store uneditable categories at the beginning of the <code>categoryLabels</code> array. For example, it's common to have an "Unfiled" category at position zero that is not editable.
<b>Result</b>	Returns <code>true</code> if any of the following conditions are <code>true</code> :	
	<ul style="list-style-type: none"><li>• The current category is renamed.</li><li>• The current category is deleted.</li><li>• The current category is merged with another category.</li></ul>	
<b>Comments</b>	You rarely call this function directly. The <code>CategorySelect</code> function calls it when the user chooses the Edit Category list item. This function both displays the Edit Categories dialog and handles the result of the user actions. It updates the <code>AppInfoType</code> structure's list of categories and reassigns database records to new categories as needed. If a user deletes a category, <code>CategoryEdit</code>	

## Categories

### Category Functions

---

moves all of the records belonging to that category to the Unfiled category. If a category is renamed to be the same as an existing category, this function moves all of the old category's records to the new category.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [CategoryEditV20](#), [CategoryEditV10](#), [DmMoveCategory](#)

## CategoryEditV20

**Purpose** Event handler for the Edit Categories dialog.

**Declared In** Category.h

**Prototype** Boolean CategoryEditV20 (DmOpenRef db,  
UInt16 \*category, UInt32 titleStrID)

**Parameters**

-> db	Database containing the categories to be edited.
<- category	Upon return, the last category selected before the dialog was closed.
-> titleStrID	The resource ID of a tSTR resource to use as the dialog's title.

**Result** Returns true if any of the following conditions are true:

- The current category is renamed.
- The current category is deleted.
- The current category is merged with another category.

**Compatibility** This function corresponds to the Palm OS 2.0 version of CategoryEdit. Implemented only if [2.0 New Feature Set](#) is present. This function is obsolete.

**See Also** [CategoryEdit](#), [CategoryEditV10](#)

## CategoryEditV10

<b>Purpose</b>	Event handler for the Edit Categories dialog.					
<b>Declared In</b>	Category.h					
<b>Prototype</b>	<code>Boolean CategoryEditV10 (DmOpenRef db,                   UInt16 *category)</code>					
<b>Parameters</b>	<table><tr><td>-&gt; db</td><td>Open database containing the categories to be edited.</td></tr><tr><td>&lt;- category</td><td>Upon return, the last category selected before the dialog was closed.</td></tr></table>		-> db	Open database containing the categories to be edited.	<- category	Upon return, the last category selected before the dialog was closed.
-> db	Open database containing the categories to be edited.					
<- category	Upon return, the last category selected before the dialog was closed.					
<b>Result</b>	<p>Returns true if any of the following conditions are true:</p> <ul style="list-style-type: none"><li>• The current category is renamed.</li><li>• The current category is deleted.</li><li>• The current category is merged with another category.</li></ul>					
<b>Compatibility</b>	This function corresponds to the Palm OS 1.0 version of CategoryEdit. It is obsolete.					
<b>See Also</b>	<a href="#">CategoryEdit</a> , <a href="#">CategoryEditV20</a>					

## CategoryFind

<b>Purpose</b>	Return the index of a category given its name.	
<b>Declared In</b>	Category.h	
<b>Prototype</b>	<code>UInt16 CategoryFind (DmOpenRef db,                   const Char *name)</code>	
<b>Parameters</b>	-> db	Open database to search.

## Categories

### Category Functions

---

-> name Category name. Pass the empty string to find the first unused category.

**Result** Returns the index of the category's entry in the `categoryLabels` array (see [AppInfoType](#)). Returns `dmAllCategories` if the category does not exist.

## CategoryFreeList

**Purpose** Unlock or free memory locked or allocated by [CategoryCreateList](#).

**Declared In** Category.h

**Prototype** `void CategoryFreeList (DmOpenRef db,  
ListType *listP, Boolean showAll,  
UInt32 editingStrID)`

**Parameters**

-> db	Open database containing the categories.
-> listP	Pointer to the category list. (See <a href="#">ListType</a> .)
-> showAll	true if the list was created with an "All" category.
-> editingStrID	The resource ID that you passed as the <code>editingStrID</code> parameter to <a href="#">CategoryCreateList</a> . This function unlocks the resource.

**Result** Returns nothing.

**Comments** You only need to call this function if you explicitly call [CategoryCreateList](#). Typical applications call `CategorySelect`, which handles both the creation and deletion of the list.

This function frees the items in the popup list `listP`'s items array and it unlocks other resources that `CategoryCreateList` may have locked.

This function does **not** remove the categories from the passed database, and it does **not** free the ListType structure pointed to by listP. (Typically, a list is freed when its form is freed.)

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

**See Also** [CategoryFreeListV10](#)

## CategoryFreeListV10

**Purpose** Unlock or free memory locked or allocated by [CategoryCreateListV10](#).

**Declared In** Category.h

**Prototype** void CategoryFreeListV10 (DmOpenRef db,  
ListType \*lst)

**Parameters** -> db Open database containing the categories.  
-> listP Pointer to the category list. (See [ListType](#).)

**Result** Returns nothing.

**Compatibility** This function corresponds to the Palm OS 1.0 version of CategoryFreeList. It is obsolete.

**See Also** [CategoryFreeList](#)

## Categories

### Category Functions

---

## CategoryGetName

**Purpose** Return the name of the specified category.

**Declared In** Category.h

**Prototype** void CategoryGetName (DmOpenRef db, UInt16 index, Char \*name)

**Parameters**

-> db	Database that contains the categories.
-> index	Category index. This is the index into the categoryLabels array in the <a href="#">AppInfoType</a> structure. You can retrieve this index from a database record's attribute word.
<- name	Buffer to hold category name. Buffer should be dmCategoryLength in size.

**Result** Stores the category name in the name buffer passed.

May display a fatal error message if the index is out of range.

**Comments** You can use this function to find out the name of a given database record's category. Use the [DmRecordInfo](#) call to obtain the category index from the given record. For example:

```
DmOpenRef myDB;  
UInt16 record, attr, category;  
Char *name;  
  
DmRecordInfo(myDB, record, &attr, NULL, NULL);  
category = attr & dmRecAttrCategoryMask;  
CategoryGetName(myDB, category, name);
```

The category's name is copied into the variable you pass for the name parameter.

**See Also** [CategorySetName](#)

## CategoryGetNext

<b>Purpose</b>	Return the index of the next category after a given category.				
<b>Declared In</b>	Category.h				
<b>Prototype</b>	<code>UInt16 CategoryGetNext (DmOpenRef db,                       UInt16 index)</code>				
<b>Parameters</b>	<table><tr><td>-&gt; db</td><td>Open database containing the categories.</td></tr><tr><td>-&gt; index</td><td>Category index.</td></tr></table>	-> db	Open database containing the categories.	-> index	Category index.
-> db	Open database containing the categories.				
-> index	Category index.				
<b>Result</b>	Category index of next category.				
<b>Comments</b>	<p>The intended use of this function is to allow your users to cycle through categories. For example, the built-in applications cycle through categories when the user presses the corresponding hard-key button. (See the <code>ListViewNextCategory</code> function in the Address Book sample application for an example.) Note that categories are not displayed in the same order as they are stored.</p> <p>Do not use this function for searching for a particular category or iterating through a category list.</p>				
<b>Compatibility</b>	<p>In Palm OS 1.0, the system chose Unfiled as one category.</p> <p>In Palm OS 2.0 and later, the system skips both Unfiled and categories with empty records.</p>				

## Categories

### Category Functions

---

## CategoryInitialize

**Purpose** Initialize the category names, IDs, and flags.

**Declared In** Category.h

**Prototype** void CategoryInitialize (AppInfoPtr appInfoP,  
UInt16 localizedAppInfoStrID)

**Parameters** ->appInfoP Pointer to the locked application info block. See [AppInfoType](#).

->localizedAppInfoStrID  
Resource ID of the localized category names.  
This must be a resource of the type  
appInfoStringsRsc ('tAIS').

**Result** Returns nothing.

**Comments** Call this function at database creation time to initialize the database's categories from a list of localized strings.

CategoryInitialize initializes the [AppInfoType](#) structure that is associated with your database. It does not create the structure. To create the structure, you must allocate it in the storage heap (using [DmNewHandle](#)) and associate it with your database using [DmSetDatabaseInfo](#).

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## CategorySelect

**Purpose** Process the selection and editing of categories.

**Declared In** Category.h

**Prototype**

```
Boolean CategorySelect (DmOpenRef db,
const FormType *frm, UInt16 ctlID, UInt16 lstID,
Boolean title, UInt16 *categoryP,
Char *categoryName, UInt8 numUneditableCategories,
UInt32 editingStrID)
```

<b>Parameters</b>		
-> db		Open database containing the categories.
-> frm		Form that contains the category popup list.
-> ctlID		ID of the popup trigger.
-> lstID		ID of the popup list.
-> title		true to have an “All” list item. (In general, if the trigger is in the form’s title bar, it should have an “All” item. If the trigger is elsewhere in the form, it should not.)
<-> categoryP		Index of the selected category. The index is the index into the categoryLabels array.
<-> categoryName		Name of the selected category.
-> numUneditableCategories		The number of categories that the user cannot edit. You should store uneditable categories at the beginning of the categoryLabels array. For example, it’s common to have an “Unfiled” category at position zero that is not editable. This function displays the uneditable categories at the end of the popup list.
-> editingStrID		The resource ID of a tSTR resource to use as the Edit Categories list item. To use the default string (“Edit Categories”), pass the constant categoryDefaultEditCategoryString.

## Categories

### *Category Functions*

---

If you don't want users to edit categories, pass the `categoryHideEditCategory` constant.

**Result** Returns `true` if any of the following conditions are true:

- The current category is renamed.
- The current category is deleted.
- The current category is merged with another category.

**Comments**

Call this function when the user taps the category popup trigger. This function handles all aspects of displaying the popup list and managing the user selection—It creates the popup list using `CategoryCreateList`, displays the popup list, calls `CategoryEdit` if the user selects the Edit Categories item, uses `CategorySetTriggerLabel` to set the trigger label to the item the user selected, and then calls `CategoryFreeList` to free the list items array. Your application is responsible for checking the value of `categoryP` upon return and updating the display or changing the record's category to the new selection.

**Compatibility**

Implemented only if [2.0 New Feature Set](#) is present.

The constants `categoryDefaultEditCategoryString` and `categoryHideEditCategory` are defined only if [3.5 New Feature Set](#) is present. In earlier versions, you can suppress the Edit Categories string by passing 0 for the `editingStrID` parameter, or include the item by passing `categoryEditStrID`.

**See Also**

[CategorySelectV10](#)

## CategorySelectV10

**Purpose** Process the selection and editing of categories.

**Declared In** Category.h

**Prototype** Boolean CategorySelectV10 (DmOpenRef db,  
const FormType \*frm, UInt16 ctlID, UInt16 lstID,  
Boolean title, UInt16 \*categoryP,  
Char \*categoryName)

**Parameters**

-> db	Open database containing the categories.
-> frm	Form that contains the category popup list.
-> ctlID	ID of the popup trigger.
-> lstID	ID of the popup list.
-> title	true to have an “All” list item. (In general, if the trigger is in the form’s title bar, it should have an “All” item. If the trigger is elsewhere in the form, it should not.)

## Categories

### Category Functions

---

<-> categoryP Index of the selected category. The index is the index into the categoryLabels array.

<-> categoryName  
Name of the selected category.

**Result** Returns true if any of the following conditions are true:

- The current category is renamed.
- The current category is deleted.
- The current category is merged with another category.

**Compatibility** This function corresponds to the Palm OS 1.0 version of [CategorySelect](#). It is obsolete.

## CategorySetName

**Purpose** Change the category name in the [AppInfoType](#) structure, or delete a category.

**Declared In** Category.h

**Prototype** void CategorySetName (DmOpenRef db, UInt16 index,  
const Char \*nameP)

**Parameters**

-> db	Open database containing the category to change.
-> index	Index of category to rename.
-> nameP	The new category name (null-terminated), or NULL to delete the category.

**Result** Returns nothing.

**Comments** The [CategoryEdit](#) function calls this function when a user creates a new category or renames an existing category in the Edit Categories dialog. Your application does not have to call it directly.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## CategorySetTriggerLabel

**Purpose** Set the label displayed by the category popup trigger.

**Declared In** Category.h

**Prototype** void CategorySetTriggerLabel (ControlType \*ctl,  
Char \*name)

**Parameters** <-> ctl                    Pointer to control object (popup trigger) to  
    relabel.  
    <-> name                    Pointer to the name of the new category.

**Result** Returns nothing.

**Comments** The CategorySetTriggerLabel function calls the [CategoryTruncateName](#) function to truncate the category name to the maximum length. The maximum length varies, depending upon which ROM is installed in the device.

---

**NOTE:** This function passes the name parameter to the CategoryTruncateName function, which means that the name value must be modifiable. CategorySetTriggerLabel does not make a copy of the string passed, so you must ensure that the string remains valid until the form is closed.

---

**See Also** [CtlsetLabel](#)

## Categories

## *Category Functions*

# CategoryTruncateName

**Purpose** Truncate a category name so that it's short enough to display. The category name is truncated if it's longer than maxWidth.

## **Declared In** Category.h

**Prototype** void CategoryTruncateName (Char \*name,  
                  UInt16 maxWidth)

<b>Parameters</b>	<-> name	Category name to truncate. Upon return, contains the truncated name.
	-> maxWidth	Maximum size, in pixels, of truncated category (including ellipsis).

**Result** Returns nothing.

# Clipboard

---

This chapter provides reference material for the clipboard API defined in `Clipboard.h`. It covers:

- [Clipboard Data Structures](#)
- [Clipboard Functions](#)

## Clipboard Data Structures

### ClipboardFormatType

The `ClipboardFormatType` enum specifies the type of data to add to the clipboard or retrieve from the clipboard.

```
enum clipboardFormats {
    clipboardText,
    clipboardInk,
    clipboardBitmap };
typedef enum clipboardFormats
    ClipboardFormatType;
```

#### Value Descriptions

<code>clipboardText</code>	Textual data. This is the most commonly used clipboard.
<code>clipboardInk</code>	Reserved.
<code>clipboardBitmap</code>	Bitmap data.

Clipboards for each type of data are separately maintained. That is, if you add a string of text to the clipboard, then add a bitmap, then ask to retrieve a `clipboardText` item from the clipboard, you will receive the string you added before the bitmap; the bitmap does not overwrite textual data and vice versa.

## Clipboard

### *Clipboard Functions*

---

# Clipboard Functions

## ClipboardAddItem

**Purpose** Add the item passed to the specified clipboard. Replaces the current item (if any) of that type.

**Declared In** Clipboard.h

**Prototype**

```
void ClipboardAddItem  
(const ClipboardFormatType format,  
 const void *ptr, UInt16 length)
```

**Parameters**

-> format	Text, ink, bitmap, etc. See <a href="#">ClipboardFormatType</a> .
-> ptr	Pointer to the item to place on the clipboard.
-> length	Size in bytes of the item to place on the clipboard.

**Result** Returns nothing.

**Comments** The clipboard makes a copy of the data that you pass to this function. Thus, you may free any data that you've passed to the clipboard without destroying the contents of the clipboard. You may also add constant data or stack-based data to the clipboard.

---

**WARNING!** You can't add null-terminated strings to the clipboard.

---

**See Also** [FldCut](#), [FldCopy](#)

## ClipboardAppendItem

<b>Purpose</b>	Append data to the item on the clipboard.	
<b>Declared In</b>	Clipboard.h	
<b>Prototype</b>	<pre>Err ClipboardAppendItem (const ClipboardFormatType format, const void *ptr, UInt16 length)</pre>	
<b>Parameters</b>	-> format	Text, ink, bitmap, etc. See <a href="#">ClipboardFormatType</a> . This function is intended to be used only for the clipboardText format.
	-> ptr	Pointer to the data to append to the item on the clipboard.
	-> length	Size in bytes of the data to append to the clipboard.
<b>Result</b>	0 upon success or memErrNotEnoughSpace if there is not enough space to append the data to the clipboard.	
<b>Comments</b>	<p>This function differs from <a href="#">ClipboardAddItem</a> in that it does not overwrite data already on the clipboard. It allows you to create a large text item on the clipboard from several small disjointed pieces. When other applications retrieve the text from the clipboard, it's retrieved as a single unit.</p> <p>This function simply appends the specified item to the item already on the clipboard without attempting to parse the format. It's assumed that you'll call it several times over a relatively short interval and that no other application will attempt to retrieve text from the clipboard before your application is finished appending.</p>	
<b>Compatibility</b>	Implemented only if <a href="#">3.2 New Feature Set</a> is present.	

## Clipboard

### Clipboard Functions

---

## ClipboardGetItem

**Purpose** Return the handle of the contents of the clipboard of a specified type and the length of a clipboard item.

**Declared In** Clipboard.h

**Prototype** MemHandle ClipboardGetItem  
(const ClipboardFormatType format, UInt16 \*length)

**Parameters** -> format      Text, ink, bitmap, etc. See  
[ClipboardFormatType](#).  
-< length      The length in bytes of the clipboard item is  
returned here.

**Result** Handle of the clipboard item.

**Comments** The handle returned is a handle to the actual clipboard chunk. It is not suitable for passing to any API that modifies memory (such as [FldSetTextHandle](#)). Consider this to be read-only access to the chunk. Copy the contents of the clipboard to your application's own storage as soon as possible and use that reference instead of the handle returned by this function.

Don't free the handle returned by this function; it is freed when a new item is added to the clipboard.

Text retrieved from the clipboard does not have a null terminator. You must use the length parameter to determine the length in bytes of the string you've retrieved.

# Controls

---

This chapter describes the control object API as declared in the header file `Control.h`. It discusses the following topics:

- [Control Data Structures](#)
- [Control Resources](#)
- [Control Functions](#)

For more information on controls, see the section “[Offscreen Windows](#)” in the *Palm OS Programmer’s Companion*, vol. I.

## Control Data Structures

### ButtonFrameType

The `ButtonFrameType` enum specifies the border style for the button. It defines values for the `frame` field of [ControlAttrType](#).

```
enum buttonFrames {noButtonFrame,  
                  standardButtonFrame, boldButtonFrame,  
                  rectangleButtonFrame};  
typedef enum buttonFrames ButtonFrameType;
```

#### Value Descriptions

<code>noButtonFrame</code>	The button has no border.
<code>standardButtonFrame</code>	Standard button rectangular border with rounded corners.
<code>boldButtonFrame</code>	Bolded rectangular border with rounded corners.
<code>rectangleButtonFrame</code>	Rectangular border with square corners.

## ControlAttrType

The ControlAttrType bit field specifies the control's visible characteristics. It is defined as follows:

```
typedef struct {
    UInt8 usable      :1;
    UInt8 enabled     :1;
    UInt8 visible     :1;
    UInt8 on          :1;
    UInt8 leftAnchor :1;
    UInt8 frame       :3;
    UInt8 drawnAsSelected : 1;
    UInt8 graphical   :1;
    UInt8 vertical    :1;
} ControlAttrType;
```

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the ControlAttrType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

Your code should treat the ControlAttrType structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

### Field Descriptions

usable	If 0, the control is not considered to be part of the interface of the current application, and it doesn't appear on screen. You can use <a href="#">CtlSetUsable</a> , <a href="#">CtlShowControl</a> , or <a href="#">CtlHideControl</a> to set or clear this value.
enabled	If 0, the control is visible but doesn't respond to the pen. This value is set by <a href="#">CtlSetEnabled</a> and returned by <a href="#">CtlEnabled</a> .

visible	Set and cleared internally when the control is drawn ( <a href="#">CtlDrawControl</a> ) and erased ( <a href="#">CtlEraseControl</a> ).
on	If set, the control has the value “on.” For example, a check box that has the on value has a check mark displayed in it. Use <a href="#">CtlGetValue</a> and <a href="#">CtlSetValue</a> to retrieve and set this value.
leftAnchor	Used by controls that expand and shrink their width when the label is changed. If this attribute is set, the left bound of the control is fixed.
frame	The type of frame drawn around the button controls. See <a href="#">ButtonFrameType</a> for possible values. Only button controls use this attribute; for all other controls, the <a href="#">ControlStyleType</a> determines the frame.
drawnAsSelected	Used on Palm OS® release 3.5 for button controls that contain no text (indicating that the button is displayed on top of a bitmap). If set, the button is drawn as inverted. If clear, the button is drawn normally.
graphical	If set, the control is a graphical control, slider, or feedback slider.
vertical	Not currently used.

## Compatibility

The drawnAsSelected, graphical, and vertical attributes are only present if [3.5 New Feature Set](#) is present.

## ControlPtr

The ControlPtr is a pointer to a [ControlType](#) structure.

## Controls

### *Control Data Structures*

---

```
typedef ControlType* ControlPtr;
```

## ControlStyleType

The `ControlStyleType` enum specifies values for the [ControlType](#) style field, which specifies the type of the control (button, push button, and so on).

```
enum controlStyles {buttonCtl, pushButtonCtl,  
    checkboxCtl, popupTriggerCtl,  
    selectorTriggerCtl, repeatingButtonCtl,  
    sliderCtl, feedbackSliderCtl};  
typedef enum controlStyles ControlStyleType;
```

### Value Descriptions

buttonCtl	Button. Buttons display a text label in a box. The <a href="#">ButtonFrameType</a> specifies the type of box.
pushButtonCtl	Push button. Selecting a push button inverts its display so that it appears highlighted.
checkboxCtl	Check box. Check boxes display a setting of either on (checked) or off (unchecked)
popupTriggerCtl	Popup trigger. Popup triggers display a graphic element followed by a text label. They are used to display popup lists.
selectorTriggerCtl	Selector trigger. Selector triggers display a text label surrounded by a gray rectangular frame. The control expands or contracts to the width of the new label.
repeatingButtonCtl	Repeating button. Repeating buttons look like buttons; however, a repeating button is repeatedly selected if the user holds the pen on it.

**sliderCtl**

Slider. Sliders display two bitmaps: one representing the current value (the thumb), and another representing the scale of available values. The user can slide the thumb to the left or the right to change the value.

**feedbackSliderCtl**

Feedback slider. A feedback slider looks like a slider; however, a feedback slider sends events each time the thumb moves while the pen is still down. A regular slider sends an event only when the user releases the pen.

**Compatibility**

The `sliderCtl` and `feedbackSliderCtl` values are only defined if [3.5 New Feature Set](#) is present.

## ControlType

The `ControlType` structure defines the type and characteristics of a control. It is defined as follows:

```
typedef struct {
    UInt16           id;
    RectangleType   bounds;
    Char *          text;
    ControlAttrType attr;
    ControlStyleType style;
    FontID           font;
    UInt8            group;
    UInt8            reserved;
} ControlType;
```

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `ControlType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

## Controls

### Control Data Structures

---

Your code should treat the `ControlType` structure as opaque. The fields in the struct are set by values you specify when you create the control resource, and they typically do not change. Use the functions specified in the descriptions below to retrieve and set the values. Do not attempt to change structure member values directly.

#### Field Descriptions

<code>id</code>	ID value you specified when you created the control resource.
<code>bounds</code>	Bounds of the control, in window-relative coordinates. The control's text label is clipped to the control's bounds. The control's frame is drawn around (outside) the bounds of the control. <a href="#">FrmGetObjectBounds</a> and <a href="#">FrmSetObjectBounds</a> retrieve and set this value.
<code>text</code>	Pointer to the control's label. If <code>text</code> is NULL, the control has no label. Use <a href="#">CtlGetLabel</a> and <a href="#">CtlSetLabel</a> to retrieve and set this value.
<code>attr</code>	Control attributes. See <a href="#">ControlAttrType</a> .
<code>style</code>	Style of the control. See <a href="#">ControlStyleType</a> .
<code>font</code>	Font to use to draw the control's label.
<code>group</code>	Group ID of a push button or a check box that is part of an exclusive group. The control routines don't automatically turn one control off when another is selected. It's up to the application or a higher-level object, like a dialog box, to manage this.
<code>reserved</code>	Reserved for future use.

#### GraphicControlType

The `GraphicControlType` struct defines a graphical control. A graphical control is like any other control except that it displays a bitmap in place of the text label.

```
typedef struct GraphicControlType {  
    UInt16           id;  
    RectangleType    bounds;  
    DmResID          bitmapID;  
    DmResID          selectedBitmapID;  
    ControlAttrType attr;  
    ControlStyleType style;  
    FontID           unused;  
    UInt8            group;  
    UInt8            reserved;  
} GraphicControlType;
```

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `GraphicControlType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

Your code should treat the `GraphicControlType` structure as opaque. The fields in the struct are set by values you specify when you create the control resource, and they typically do not change. Use the functions specified in the descriptions below to retrieve and set the values. Do not attempt to change structure member values directly.

### Field Descriptions

<code>id</code>	ID value you specified when you created the control resource.
<code>bounds</code>	Bounds of the control, in window-relative coordinates. The control's frame is drawn around (outside) the bounds of the control. <a href="#">FrmGetObjectBounds</a> and <a href="#">FrmSetObjectBounds</a> retrieve and set this value.
<code>bitmapID</code>	Resource ID of the bitmap to display in the button. You can use <a href="#">CtlSetGraphics</a> to change this value.

## Controls

### Control Data Structures

---

selectedBitmapID	If the button should show a different bitmap when selected, this field contains the resource ID of that bitmap. You typically use this field for push buttons or repeating buttons. <a href="#">CtlSetGraphics</a> can change this value.
attr	Control attributes. See <a href="#">ControlAttrType</a> . For a graphical control, the graphical attribute must be set. The APIs described in the ControlAttrType section can be used to access the bitfields here. Because the ControlAttrType APIs take a ControlType* as an argument, the GraphicControlType* should be cast to a ControlType* when making the API calls.
style	Style of the control. See <a href="#">ControlStyleType</a> . A graphical control can be any type of control other than checkboxCtl.
unused	Unused.
group	Group ID of a push button that is part of an exclusive group. The control routines don't automatically turn one control off when another is selected. It's up to the application or a higher-level object, like a dialog box, to manage this.
reserved	Reserved for future use.

**Compatibility** This struct is defined only if [3.5 New Feature Set](#) is present.

## SliderControlType

The SliderControlType struct defines a slider control or a feedback slider control.

```
typedef struct SliderControlType {
    UInt16 id;
    RectangleType bounds;
    DmResID thumbID;
    DmResID backgroundID;
    ControlAttrType attr;
    ControlStyleType style;
    UInt8 reserved;
    Int16 minValue;
    Int16 maxValue;
    Int16 pageSize;
    Int16 value;
    MemPtr activeSliderP;
} SliderControlType;
```

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `SliderControlType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

Your code should treat the `SliderControlType` structure as opaque. The fields in the struct are set by values you specify when you create the control resource, and they typically do not change. You can use [CtlSetSliderValues](#) to set new minimum, maximum, page size, and current values, and [CtlGetSliderValues](#) to retrieve these values. Do not attempt to change structure member values directly.

### Field Descriptions

<code>id</code>	ID value you specified when you created the control resource.
<code>bounds</code>	Bounds of the control, in window-relative coordinates. <a href="#">FrmGetObjectBounds</a> and <a href="#">FrmSetObjectBounds</a> retrieve and set this value.

## Controls

### Control Data Structures

---

thumbID	Resource ID of the bitmap to use for the slider knob (called the “thumb”). If NULL, the default bitmap is used.
backgroundID	Resource ID of the bitmap to use for the slider background. If NULL, the default bitmap is used.
attr	Control attributes. See <a href="#">ControlAttrType</a> . For a slider, the graphical attribute is set. The APIs described in the ControlAttrType section can be used to access the bitfields here. Because the ControlAttrType APIs take a ControlType* as an argument, the SliderControlType* should be cast to a ControlType* when making the API calls.
style	Style of the control. See <a href="#">ControlStyleType</a> . Must be sliderCtl or feedbackSliderCtl.
reserved	Reserved for future use.
minValue	Value of the slider when the thumb is all the way to the left.
maxValue	Value of the slider when the thumb is all the way to the right.
pageSize	Amount by which to increase or decrease the slider value when the user taps to the right or left of the thumb.
value	Current value represented by the slider. Use <a href="#">CtlGetValue</a> and <a href="#">CtlSetValue</a> to retrieve and set this value.
activeSliderP	Pointer to a memory location used when the slider is active. A slider is active if it is currently being drawn or if it is tracking the pen. If the slider is inactive, this pointer is NULL.

**Compatibility** This struct is defined only if [3.5 New Feature Set](#) is present.

## Control Resources

Different resources are associated with different controls, as follows:

- Button—[Button Resource](#) (tBTN)
- Popup trigger—[Popup Trigger Resource](#) (tPUT)
- Selector trigger—[Selector Trigger Resource](#) (tSLT)
- Repeat control—[Repeating Button Resource](#) (tREP)
- Push button—[Push Button Resource](#) (tPBN)
- Check box—[Check Box Resource](#) (tCBX)
- Slider—Slider Resource (tsld)
- Feedback slider—Feedback Slider Resource (tslf)

## Control Functions

### CtlDrawControl

<b>Purpose</b>	Draw a control object (and the text or graphic in it) on screen.
<b>Declared In</b>	Control.h
<b>Prototype</b>	void CtlDrawControl (ControlType *controlP)
<b>Parameters</b>	-> controlP      Pointer to the control object to draw. (See <a href="#">ControlType</a> .)
<b>Result</b>	Returns nothing.
<b>Comments</b>	The control is drawn only if its <code>usable</code> attribute is <code>true</code> . This function sets the <code>visible</code> attribute to <code>true</code> .
<b>Compatibility</b>	In releases prior to Palm OS® 3.5, it is common to create graphical buttons by drawing a button with no text label on top of a bitmap. If <a href="#">3.5 New Feature Set</a> is present, you should use graphical controls instead. (See <a href="#">GraphicControlType</a> .) CtlDrawControl attempts

## Controls

### *Control Functions*

---

to provide backward compatibility for the old-style graphical buttons.

**See Also** [CtlSetUsable](#), [CtlShowControl](#)

## CtlEnabled

**Purpose** Return `true` if the control responds to the pen.

**Declared In** Control.h

**Prototype** Boolean CtlEnabled (const ControlType \*controlP)

**Parameters** -> controlP Pointer to control object. (See [ControlType](#).)

**Result** Returns `true` if the controls object responds to the pen; `false` if not.

**Comments** This function provides no indication of whether the control is visible on the screen. A control that doesn't respond to the pen may be visible, and if so, its appearance is no different from controls that do respond to the pen. You might use such a control to display some state of your application that cannot be modified.

**See Also** [CtlsetEnabled](#)

## CtlEraseControl

**Purpose** Erase a usable and visible control object and its frame from the screen.

**Declared In** Control.h

**Prototype** void CtlEraseControl (ControlType \*controlP)

**Parameters** -> controlP Pointer to control object to erase. (See [ControlType](#).)

<b>Comments</b>	This function sets the <code>visible</code> attribute to <code>false</code> . If <a href="#">3.5 New Feature Set</a> is present, it also sets the <code>drawnAsSelected</code> attribute to <code>false</code> .  Don't call this function directly; instead, use <a href="#"><code>FrmHideObject</code></a> , which calls this function.
-----------------	---

## CtlGetLabel

<b>Purpose</b>	Return a character pointer to a control's text label.
<b>Declared In</b>	<code>Control.h</code>
<b>Prototype</b>	<code>const Char *CtlGetLabel (const ControlType *controlP)</code>
<b>Parameters</b>	<code>-&gt; controlP</code> Pointer to control object. (See <a href="#"><code>ControlType</code></a> .)
<b>Result</b>	Returns a pointer to a null-terminated string.
<b>Comments</b>	Make sure that <code>controlP</code> is not a graphical control or a slider control. The graphical control and slider control structures do not contain a text label field.
<b>See Also</b>	<a href="#"><code>CtlSetLabel</code></a>

## CtlGetSliderValues

<b>Purpose</b>	Return current values used by a slider control.
<b>Declared In</b>	<code>Control.h</code>
<b>Prototype</b>	<code>void CtlGetSliderValues (const ControlType *ctlP, UIInt16 *minValueP, UInt16 *maxValueP, UIInt16 *pageSizeP, UInt16 *valueP)</code>
<b>Parameters</b>	<code>-&gt; ctlP</code> Pointer to a control object. (See <a href="#"><code>ControlType</code></a> .)

## Controls

### Control Functions

---

<- minValueP	The slider's minimum value. Pass NULL if you don't want to retrieve this value.
<- maxValueP	The slider's maximum value. Pass NULL if you don't want to retrieve this value.
<- pageSizeP	The slider's page size value. Pass NULL if you don't want to retrieve this value.
<- valueP	The slider's current value. Pass NULL if you don't want to retrieve this value.

**Result** Returns nothing. The slider's values are returned in the parameters to this function.

**Comments** If `ctlP` is not a slider or a feedback slider, this function immediately returns.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [CtlSetSliderValues](#), [SliderControlType](#)

## CtlGetValue

**Purpose** Return the current value of the specified control.

**Declared In** Control.h

**Prototype** Int16 CtlGetValue (const ControlType \*controlP)

**Parameters** -> controlP Pointer to a control object. (See [ControlType](#).)

**Result** Returns the current value of the control. For most controls the return value is either 0 (off) or 1 (on). For sliders, this function returns the value of the `value` field.

**See Also** [CtlSetValue](#), [FrmGetControlGroupSelection](#),  
[FrmSetControlGroupSelection](#), [FrmGetControlValue](#),  
[FrmSetControlValue](#)

## CtlHandleEvent

**Purpose** Handle event in the specified control object.

**Declared In** Control.h

**Prototype** Boolean CtlHandleEvent (ControlType \*controlP,  
EventType \*pEvent)

**Parameters**

-> controlP	Pointer to control object. (See <a href="#">ControlType</a> .)
-> pEvent	Pointer to an EventType structure.

**Result** Returns true if an event is handled by this function. Events that are handled are:

- [penDownEvent](#) — If the pen is within the bounds of the control
- [ctlEnterEvent](#), [ctlRepeatEvent](#), and [ctlExitEvent](#) — If the control ID in the event data matches the control's ID.

**Comments** The control object must be usable, visible, and respond to the pen for this function to handle the event.

When this routine receives a penDownEvent, it checks if the pen position is within the bounds of the control object. If it is, a ctlEnterEvent is added to the event queue and the routine exits.

When this routine receives a ctlEnterEvent, the control object is redrawn as necessary as either selected or deselected, depending on its previous state.

When this routine receives a ctlEnterEvent or ctlRepeatEvent, it checks that the control ID in the passed event record matches the ID of the specified control. If they match, this routine tracks the pen until it comes up or until it leaves the object's bounds. When that happens, ctlSelectEvent is sent to the event queue if the pen came up in the bounds of the control. If the pen exits the bounds, a ctlExitEvent is sent to the event queue.

## Controls

### *Control Functions*

---

## CtlHideControl

**Purpose** Set a control's usable attribute to `false` and erase the control from the screen.

**Declared In** Control.h

**Prototype** void CtlHideControl (ControlType \*controlP)

**Parameters** -> controlP Pointer to the control object to hide. (See [ControlType](#).)

**Result** Returns nothing.

**Comments** A control that is not usable doesn't draw and doesn't respond to the pen.

This function is the same as [CtlEraseControl](#) except that it also sets `usable` to `false` (in addition to setting `visible` to `false`).

Don't call this function directly; instead, use [FrmHideObject](#), which performs the same function and works for all user interface objects.

**See Also** [CtlShowControl](#)

## CtlHitControl

**Purpose** Simulate tapping a control. This function adds a [ctlSelectEvent](#) to the event queue.

**Declared In** Control.h

**Prototype** void CtlHitControl (const ControlType \*controlP)

**Parameters** -> controlP Pointer to a control object. (See [ControlType](#).)

**Result** Returns nothing.

**Comments** Useful for testing.

## CtlNewControl

**Purpose** Create a new control object dynamically and install it in the specified form.

**Declared In** Control.h

**Prototype**

```
ControlType *CtlNewControl (void **formPP,  
                           UInt16 ID, ControlStyleType style,  
                           const Char *textP, Coord x, Coord y, Coord width,  
                           Coord height, FontID font, UInt8 group,  
                           Boolean leftAnchor)
```

**Parameters**

<-> formPP	Pointer to the pointer to the form in which the new control is installed. This value is not a handle; that is, the formPP value may change if the object moves in memory. In subsequent calls, always use the new formPP value returned by this function.
-> ID	Symbolic ID of the control.
-> style	A <a href="#">ControlStyleType</a> value specifying the kind of control to create: button, push button, repeating button, check box, popup trigger, or popup selector. To create a graphical control or slider control dynamically, use <a href="#">CtlNewGraphicControl</a> or <a href="#">CtlNewSliderControl</a> , respectively.

## Controls

### *Control Functions*

---

-> textP	Pointer to the control's label text. If <code>textP</code> is <code>NULL</code> , the control has no label. Only buttons, push buttons, and text boxes have text labels. Because the contents of this pointer are copied into their own buffer, you can free the <code>textP</code> pointer any time after the <code>CtlNewControl</code> function returns. The buffer into which this string is copied is freed automatically when you remove the control from the form or delete the form.
-> x	Horizontal coordinate of the upper-left corner of the control's boundaries, relative to the window in which it appears.
-> y	Vertical coordinate of the upper-left corner of the control's boundaries, relative to the window in which it appears.
-> width	Width of the control, expressed in pixels. Valid values are 1–160. If the value of either of the <code>width</code> or <code>height</code> parameters is 0, the control is sized automatically as necessary to display the text passed as the value of the <code>text</code> parameter.
-> height	Height of the control, expressed in pixels. Valid values are 1–160. If the value of either of the <code>width</code> or <code>height</code> parameters is 0, the control is sized automatically as necessary to display the text passed as the value of the <code>text</code> parameter.
-> font	Font used to draw the control's label.
-> group	Group ID of a push button or a check box that is part of an exclusive group. The control routines don't turn one control off automatically when another is selected. It's up to the application or a higher-level object, such as a dialog box, to manage this.

-> leftAnchor true specifies that the left bound of this control is fixed. This attribute is used by controls that resize dynamically in response to label text changes.

**Result** Returns a pointer to the new control.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [CtlValidatePointer](#), [FrmRemoveObject](#)

## CtlNewGraphicControl

**Purpose** Create a new graphical control dynamically and install it in the specified form.

**Declared In** Control.h

**Prototype** GraphicControlType \*CtlNewGraphicControl  
(void \*\*formPP, UInt16 ID,  
ControlStyleType style, DmResID bitmapID,  
DmResID selectedBitmapID, Coord x, Coord y,  
Coord width, Coord height, UInt8 group,  
Boolean leftAnchor)

**Parameters** <-> formPP Pointer to the pointer to the form in which the new control is installed. This value is not a handle; that is, the formPP value may change if the object moves in memory. In subsequent calls, always use the new formPP value returned by this function.

-> ID Symbolic ID of the control.

-> style A [ControlStyleType](#) value specifying the kind of control to create: button, push button, popup trigger, repeating button, or popup selector. Graphic controls cannot be check boxes.

## Controls

### *Control Functions*

---

-> bitmapID	Resource ID of the bitmap to display on the control.
-> selectedBitmapID	Resource ID of the bitmap to display when the control is selected, if different from bitmapID.
-> x	Horizontal coordinate of the upper-left corner of the control's boundaries, relative to the window in which it appears.
-> y	Vertical coordinate of the upper-left corner of the control's boundaries, relative to the window in which it appears.
-> width	Width of the control, expressed in pixels. Valid values are 1–160.
-> height	Height of the control, expressed in pixels. Valid values are 1–160.
-> group	Group ID of a push button that is part of an exclusive group. The control routines don't turn one control off automatically when another is selected. It's up to the application or a higher-level object, such as a dialog box, to manage this.
-> leftAnchor	true specifies that the left bound of this control is fixed.

**Result** Returns a pointer to the new graphical control. See [GraphicControlType](#).

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [CtlNewSliderControl](#), [CtlNewControl](#),  
[CtlValidatePointer](#), [FrmRemoveObject](#)

## CtINewSliderControl

<b>Purpose</b>	Create a new slider or feedback slider dynamically and install it in the specified form.												
<b>Declared In</b>	<code>Control.h</code>												
<b>Prototype</b>	<pre>SliderControlType *CtINewSliderControl (void **formPP, UInt16 ID, ControlStyleType style, DmResID thumbID, DmResID backgroundID, Coord x, Coord y, Coord width, Coord height, UInt16 minValue, UInt16 maxValue, UInt16 pageSize, UInt16 value)</pre>												
<b>Parameters</b>	<table><tr><td>&lt;-&gt; <code>formPP</code></td><td>Pointer to the pointer to the form in which the new control is installed. This value is not a handle; that is, the <code>formPP</code> value may change if the object moves in memory. In subsequent calls, always use the new <code>formPP</code> value returned by this function.</td></tr><tr><td>-&gt; <code>ID</code></td><td>Symbolic ID of the slider.</td></tr><tr><td>-&gt; <code>style</code></td><td>Either <code>sliderCtl</code> or <code>feedbackSliderCtl</code>. See <a href="#">ControlStyleType</a>.</td></tr><tr><td>-&gt; <code>thumbID</code></td><td>Resource ID of the bitmap to display as the slider thumb. The slider thumb is the knob that the user can drag to change the slider's value. To use the default thumb bitmap, pass <code>NULL</code> for this parameter.</td></tr><tr><td>-&gt; <code>backgroundID</code></td><td>Resource ID of the bitmap to display as the slider background. To use the default background bitmap, pass <code>NULL</code> for this parameter.</td></tr><tr><td>-&gt; <code>x</code></td><td>Horizontal coordinate of the upper-left corner of the slider's boundaries, relative to the window in which it appears.</td></tr></table>	<-> <code>formPP</code>	Pointer to the pointer to the form in which the new control is installed. This value is not a handle; that is, the <code>formPP</code> value may change if the object moves in memory. In subsequent calls, always use the new <code>formPP</code> value returned by this function.	-> <code>ID</code>	Symbolic ID of the slider.	-> <code>style</code>	Either <code>sliderCtl</code> or <code>feedbackSliderCtl</code> . See <a href="#">ControlStyleType</a> .	-> <code>thumbID</code>	Resource ID of the bitmap to display as the slider thumb. The slider thumb is the knob that the user can drag to change the slider's value. To use the default thumb bitmap, pass <code>NULL</code> for this parameter.	-> <code>backgroundID</code>	Resource ID of the bitmap to display as the slider background. To use the default background bitmap, pass <code>NULL</code> for this parameter.	-> <code>x</code>	Horizontal coordinate of the upper-left corner of the slider's boundaries, relative to the window in which it appears.
<-> <code>formPP</code>	Pointer to the pointer to the form in which the new control is installed. This value is not a handle; that is, the <code>formPP</code> value may change if the object moves in memory. In subsequent calls, always use the new <code>formPP</code> value returned by this function.												
-> <code>ID</code>	Symbolic ID of the slider.												
-> <code>style</code>	Either <code>sliderCtl</code> or <code>feedbackSliderCtl</code> . See <a href="#">ControlStyleType</a> .												
-> <code>thumbID</code>	Resource ID of the bitmap to display as the slider thumb. The slider thumb is the knob that the user can drag to change the slider's value. To use the default thumb bitmap, pass <code>NULL</code> for this parameter.												
-> <code>backgroundID</code>	Resource ID of the bitmap to display as the slider background. To use the default background bitmap, pass <code>NULL</code> for this parameter.												
-> <code>x</code>	Horizontal coordinate of the upper-left corner of the slider's boundaries, relative to the window in which it appears.												

## Controls

### Control Functions

---

-> y	Vertical coordinate of the upper-left corner of the slider's boundaries, relative to the window in which it appears.
-> width	Width of the slider, expressed in pixels. Valid values are 1–160.
-> height	Height of the slider, expressed in pixels. Valid values are 1–160.
-> minValue	Value of the slider when its thumb is all the way to the left.
-> maxValue	Value of the slider when its thumb is all the way to the right.
-> pageSize	Amount by which to increase or decrease the slider's value when the user clicks to the right or left of the thumb.
-> value	The initial value to display in the slider.

**Result** Returns a pointer to the new slider control. See [SliderControlType](#).

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [CtlNewGraphicControl](#), [CtlNewControl](#), [CtlValidatePointer](#), [FrmRemoveObject](#)

## CtlsetEnabled

**Purpose** Set a control as enabled or disabled. Disabled controls do not respond to the pen.

**Declared In** Control.h

**Prototype** void CtlsetEnabled (ControlType \*controlP,  
Boolean usable)

**Parameters** -> controlP Pointer to a control object. (See [ControlType](#).)

-> usable      true to enable the control; false to disable the control.

**Result** Returns nothing.

**Comments** If you disable a visible control, the control is still displayed, and its appearance is no different from controls that do respond to the pen. You might use such a control to inform your users of some state of your application that cannot be modified.

**See Also** [CtlEnabled](#)

## CtlSetGraphics

**Purpose** Set the bitmaps for a graphical control and redraw the control if it is visible.

**Declared In** Control.h

**Prototype** void CtlSetGraphics (ControlType \*ctlP,  
DmResID newBitmapID, DmResID newSelectedBitmapID)

**Parameters**

-> ctlP	Pointer to a graphical control object. (See <a href="#">GraphicControlType</a> .)
-> newBitmapID	Resource ID of a new bitmap to display on the control, or NULL to use the current bitmap.
-> newSelectedBitmapID	Resource ID of a new bitmap to display when the control is selected, or NULL to use the current selected bitmap.

**Result** Returns nothing.

**Comments** If ctlP is not a graphical control, this function immediately returns.

## Controls

### *Control Functions*

---

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [GraphicControlType](#)

## CtlsetLabel

**Purpose** Set the current label for the specified control object and redraw the control if it is visible.

**Declared In** Control.h

**Prototype** void CtlsetLabel (ControlType \*controlP,  
const Char \*newLabel)

**Parameters** -> controlP Pointer to a control object. (See [ControlType](#).)  
-> newLabel Pointer to the new text label. Must be a null-terminated string.

**Result** Returns nothing.

**Comments** This function resizes the width of the control to the size of the new label.

This function stores the newLabel pointer in the control's data structure. It doesn't make a copy of the string that is passed in. Therefore, if you use CtlsetLabel, you must manage the string yourself. You must ensure that it persists for as long as it is being displayed (that is, for as long as the control is displayed or until you call CtlsetLabel with a new string), and you must free the string after it is no longer in use (typically after the form containing the control is freed).

If you never use CtlsetLabel, you do not need to worry about freeing a control's label.

Make sure that controlP is not a graphical control or a slider control. The graphical controls and slider control structures do not

contain a text label field, so attempting to set one will crash your application.

**See Also** [CtlGetLabel](#)

## CtlSetSliderValues

**Purpose** Change a slider control's values and redraw the slider if it is visible.

**Declared In** Control.h

**Prototype** void CtlSetSliderValues (ControlType \*ctlP,  
const UInt16 \*minValueP, const UInt16 \*maxValueP,  
const UInt16 \*pageSizeP, const UInt16 \*valueP)

**Parameters**

-> ctlP	Pointer to an inactive slider or feedback slider control. (See <a href="#">SliderControlType</a> .)
-> minValueP	Pointer to a new value to use for the slider's minimum, or NULL if you don't want to change this value.
-> maxValueP	Pointer to a new value to use for the slider's maximum, or NULL if you don't want to change this value.
-> pageSizeP	Pointer to a new value to use for the slider's page size, or NULL if you don't want to change this value.
-> valueP	Pointer to a new value to use for the current value, or NULL if you don't want to change this value.

**Result** Returns nothing.

**Comments** The control's style must be `sliderCtl` or `feedbackSliderCtl`, and it not be currently tracking the pen. If the slider is currently tracking the pen, use [CtlSetValue](#) to set the `value` field.

## Controls

### *Control Functions*

---

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [CtlGetSliderValues](#), [SliderControlType](#)

## CtlSetUsable

**Purpose** Set a control to usable or not usable.

**Declared In** Control.h

**Prototype** void CtlSetUsable (ControlType \*controlP,  
Boolean usable)

**Parameters** -> controlP Pointer to a control object. (See [ControlType](#).)  
-> usable true to have the control be usable; false to have the control be not usable.

**Result** Returns nothing.

**Comments** A control that is not usable doesn't draw and doesn't respond to the pen.

This function doesn't usually update the control.

**See Also** [CtlEraseControl](#), [CtlHideControl](#), [CtlShowControl](#)

## CtlSetValue

**Purpose** Set the current value of the specified control. If the control is visible, it's redrawn.

**Declared In** Control.h

**Prototype** void CtlSetValue (ControlType \*controlP,  
Int16 newValue)

**Parameters** -> controlP Pointer to a control object. (See [ControlType](#).)

-> newValue      New value to set for the control. For sliders, specify a value between the slider's minimum and maximum. For graphical controls, push buttons, or check boxes, specify 0 for off, nonzero for on.

**Result**      Returns nothing.

**Comments**      This function works only with graphical controls, sliders, push buttons, and check boxes. If you set the value of any other type of control, the behavior is undefined.

**Compatibility**      Sliders and graphical controls are only supported if [3.5 New Feature Set](#) is present.

**See Also**      [CtlGetValue](#), [FrmGetControlGroupSelection](#),  
[FrmSetControlGroupSelection](#), [FrmGetControlValue](#),  
[FrmSetControlValue](#)

## CtlShowControl

**Purpose**      Set a control's usable attribute to true and draw the control on the screen. This function calls [CtlDrawControl](#).

**Declared In**      Control.h

**Prototype**      void CtlShowControl (ControlType \*controlP)

**Parameters**      -> controlP      Pointer to a control object. (See [ControlType](#).)

**Result**      Returns nothing.

**Comments**      If the control is already usable, this function is the functional equivalent of [CtlDrawControl](#).

Sets the visible and the usable attributes to true. (See [ControlAttrType](#).)

## Controls

### *Control Functions*

---

Don't use this function directly; instead use [FrmShowObject](#), which does the same thing.

**See Also** [CtlHideControl](#)

## CtlValidatePointer

**Purpose** Returns true if the specified pointer references a valid control object.

**Declared In** Control.h

**Prototype** Boolean CtlValidatePointer  
(const ControlType \*controlP)

**Parameters** -> controlP Pointer to a control. (See [ControlType](#).)

**Result** Returns true when passed a valid pointer to a control; otherwise, returns false.

**Comments** For debugging purposes; do not include this function in commercial products. In debug builds, this function displays a dialog and waits for the debugger when an error occurs.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [FrmValidatePtr](#), [WinValidateHandle](#)

# Date and Time Selector

---

The Palm OS® UI provides two system resources for accepting date and time input values. These resources are dialog boxes that contain UI gadgetry for entering dates and times. The Palm OS UI also provides routines to manage the interaction with these resources. This chapter describes those functions.

The API described in this chapter is declared in the header files `Day.h`, `SelDay.h`, `SelTime.h`, and `SelTimeZone.h`.

## Date and Time Selections Data Structures

### SelectDayType

```
typedef enum {
    selectDayByDay,      // return d/m/y
    selectDayByWeek,     // return d/m/y with d as
                         // same day of the week
    selectDayByMonth // return d/m/y with d as
                      // same day of the month
} SelectDayType;
```

### DaySelectorType

```
typedef struct DaySelectorType {
    RectangleType bounds;
    Boolean visible;
    UInt8 reserved1;
    Int16 visibleMonth; // month actually
                        // displayed
```

## Date and Time Selector

### Date and Time Selection Functions

---

```
    Int16          visibleYear; // year actually  
                           // displayed  
    DateTimeType   selected;  
    SelectDayType  selectDayBy;  
    UInt8          reserved2;  
} DaySelectorType;
```

## HMSTime

```
typedef struct {  
    UInt8 hours;  
    UInt8 minutes;  
    UInt8 seconds;  
    UInt8 reserved;  
} HMSTime;
```

# Date and Time Selection Functions

## DayDrawDays

**Purpose** Draw only the days-of-the-month portion of a day selector control object.

**Declared In** Day.h

**Prototype** void DayDrawDays  
(const DaySelectorType \*selectorP)

**Parameters** -> selectorP Pointer to the control object to draw.

**Result** Nothing.

**Comments** This function is used when the year or month changes. Only drawing the portion of the control that presents the days of the month avoids the flicker that would occur if the week titles were redrawn.

**Compatibility** If [5.0 New Feature Set](#) is present this function is unimplemented.

**See Also** [DayDrawDaySelector](#)

## DayDrawDaySelector

**Purpose** Draw a day selector control object on screen.

**Declared In** Day.h

**Prototype** void DayDrawDaySelector  
(const DaySelectorType \*selectorP)

**Parameters** -> selectorP Pointer to the control object to draw.

**Result** Nothing.

**Comments** The control is drawn only if it is usable.

**Compatibility** If [5.0 New Feature Set](#) is present this function is unimplemented.

**See Also** [DayDrawDays](#)

## DayHandleEvent

**Purpose** Handle event in the specified control. This routine handles two types of events, [penDownEvent](#) and [ctlEnterEvent](#).

**Declared In** Day.h

**Prototype** Boolean DayHandleEvent  
(DaySelectorType \*selectorP,  
const EventType \*pEvent)

**Parameters** -> selectorP Pointer to control object.

## Date and Time Selector

### Date and Time Selection Functions

---

-> pEvent      Pointer to an EventType structure.

**Result** true if the event was handled or false if it was not.

Posts a [daySelectEvent](#) with information on whether to use the date.

**Comments** A date is used if the user selects a day in the visible month.

When this routine receives a [penDownEvent](#), it checks if the pen position is within the bounds of the control object. If it is, a dayEnterEvent is added to the event queue and the routine exits.

When this routine receives a dayEnterEvent, it checks that the control id in the event record matches the id of the control specified. If they match, this routine will track the pen until it comes up in the bounds in which case daySelectEvent is sent.

If the pen exits the bounds a dayExitEvent is sent.

## SelectDay

**Purpose** Display a form showing a date; allow user to select a different date.

**Declared In** SelDay.h

**Prototype** Boolean SelectDay  
(const SelectDayType selectDayBy, Int16 \*month,  
Int16 \*day, Int16 \*year, const Char \*title)

**Parameters** selectDayBy      The method by which the user should choose the day. Possible values are [selectDayByDay](#), [selectDayByWeek](#), and [selectDayByMonth](#). See [SelectDayType](#)

<-> month, day, year  
Month, day, and year selected.

-> title      String title for the dialog.

**Result** true if the OK button was pressed. If true, month, day, and year contain the new date.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

**See Also** [SelectDayV10](#)

## SelectDayV10

**Purpose** Display a form showing a date, allow user to select a different date.

**Declared In** SelDay.h

**Prototype** Boolean SelectDayV10 (Int16 \*month, Int16 \*day, Int16 \*year, const Char \*title)

**Parameters** <-> month, day, year  
Month, day, and year selected. The initial values passed in these parameters must be valid.

-> title String title for the dialog.

**Result** Returns true if the OK button was pressed. In that case, the parameters passed are changed.

**Compatibility** This function corresponds to the 1.0 version of SelectDay.

**See Also** [SelectDay](#)

## SelectOneTime

**Purpose** Display a form showing the time and allow the user to select a different time.

**Declared In** SelTime.h

**Prototype** Boolean SelectOneTime (Int16 \*hour, Int16 \*minute, const Char \*titleP)

**Parameters** <-> hour The hour selected in the form.

## Date and Time Selector

### Date and Time Selection Functions

---

<-> minute	The minute selected in the form.
-> titleP	A pointer to a string to display as the title. Doesn't change as the function executes.
<b>Result</b>	Returns <code>true</code> if the user selects OK and <code>false</code> otherwise. If <code>true</code> is returned, the values in <code>hour</code> and <code>minute</code> have probably been changed.
<b>Comments</b>	Use this function instead of <code>SelectTime</code> if you want to display a dialog that specifies a single point in time, not a range of time from start to end.
<b>Compatibility</b>	Implemented only if <a href="#">3.1 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">SelectTimeV33</a>

## SelectTime

<b>Purpose</b>	Display a form showing a start and end time. Allow the user to select a different time.				
<b>Declared In</b>	<code>SelTime.h</code>				
<b>Prototype</b>	<pre>Boolean SelectTime (TimeType *startTimeP, TimeType * endTimeP, Boolean untimed, const Char *titleP, Int16 startOfDay, Int16 endOfDay, Int16 startOfDisplay)</pre>				
<b>Parameters</b>	<table><tr><td>&lt;-&gt; startTimeP, endTimeP</td><td>Pointers to values of type <code>TimeType</code>. Pass values to display in these two parameters. If the user makes a selection and taps the OK button, the selected values are returned here.</td></tr><tr><td>-&gt; untimed</td><td>Pass in <code>true</code> to indicate that no time is selected. If the user sets the time to no time then <code>startTimeP</code> and <code>endTimeP</code> are both set to the constant <code>noTime</code> (-1).</td></tr></table>	<-> startTimeP, endTimeP	Pointers to values of type <code>TimeType</code> . Pass values to display in these two parameters. If the user makes a selection and taps the OK button, the selected values are returned here.	-> untimed	Pass in <code>true</code> to indicate that no time is selected. If the user sets the time to no time then <code>startTimeP</code> and <code>endTimeP</code> are both set to the constant <code>noTime</code> (-1).
<-> startTimeP, endTimeP	Pointers to values of type <code>TimeType</code> . Pass values to display in these two parameters. If the user makes a selection and taps the OK button, the selected values are returned here.				
-> untimed	Pass in <code>true</code> to indicate that no time is selected. If the user sets the time to no time then <code>startTimeP</code> and <code>endTimeP</code> are both set to the constant <code>noTime</code> (-1).				

-> titleP	A pointer to a string to display as the title. Doesn't change as the function executes.
-> startOfDay	The hour that the hour list displays at its top. To see earlier hours, the user can scroll the list up. The value must be between 0 to 12, inclusive.
-> endOfDay	The hour used when the "All Day" button is selected.
-> startOfDisplay	First hour initially visible.

**Result** Returns `true` if the user selects OK and `false` otherwise. If `true` is returned, the values in `hour` and `minute` have probably been changed.

**Comments** This version of `SelectTime` adds the `endOfDay` and `startOfDisplay` functionality.

**Compatibility** Implemented if [3.5 New Feature Set](#) is present.

**See Also** [SelectDay](#), [SelectOneTime](#)

## SelectTimeV33

**Purpose** Display a form showing the time and allow the user to select a different time.

This function is obsolete and should not be used.

## Date and Time Selector

### Date and Time Selection Functions

---

**Declared In** SelTime.h

**Prototype** Boolean SelectTimeV33 (TimeType \*startTimeP,  
TimeType \*endTimeP, Boolean untimed,  
const Char \*titleP, Int16 startOfDay)

**Parameters**

<-> startTimeP, endTimeP	Pointers to values of type TimeType. Pass values to display in these two parameters. If the user makes a selection and taps the OK button, the selected values are returned here.
-> untimed	Pass in true to indicate that no time is selected. If the user sets the time to no time then startTimeP and endTimeP are both set to the constant noTime (-1).
-> titleP	A pointer to a string to display as the title. Doesn't change as the function executes.
-> startOfDay	The hour that the hour list displays at its top. To see earlier hours, the user can scroll the list up. The value must be between 0 to 12, inclusive.

**Result** Returns true if the user selects OK and false otherwise. If true is returned, the values in hour and minute have probably been changed.

---

**Comments** **NOTE:** Obsolete functions are provided ONLY for backward compatibility; for example, so a 1.0 application will work on 3.x OS releases. New code should not call these routines!

---

**See Also** [SelectDay](#), [SelectOneTime](#)

## SelectTimeZone

<b>Purpose</b>	Display a form that allows the user to select a different time zone.
<b>Declared In</b>	SelTimeZone.h
<b>Prototype</b>	<pre>Boolean SelectTimeZone (Int16 *ioTimeZoneP, LmLocaleType *ioLocaleInTimeZoneP, const Char *titleP, Boolean showTimes, Boolean anyLocale)</pre>
<b>Parameters</b>	<p>&lt;-&gt; ioTimeZoneP A pointer to the time zone, given as minutes east of Greenwich Mean Time (GMT). The initial value is used as the initial selection in the form. Upon return, this parameter contains a pointer to the new time zone that the user selected.</p> <p>&lt;-&gt; ioLocaleInTimeZoneP A pointer to a locale (see <a href="#">LmLocaleType</a>) that identifies the time zone country. This parameter is used for countries that share a time zone, such as Canada and Chile.</p> <p>-&gt; titleP If the time zone specified by <code>ioTimeZoneP</code> is specific to a particular country, you do not have to initialize this parameter. Instead, set the <code>anyLocale</code> parameter to <code>true</code> to have this parameter ignored upon entry.</p> <p>-&gt; showTimes A string to use as the title for the dialog. Pass <code>NULL</code> to use the default title, which is "Set Time Zone".</p> <p>-&gt; showTimes If <code>true</code>, the dialog shows the correct times in both the current and newly selected time zones. If <code>false</code>, the dialog doesn't show the current time. Using <code>false</code> provides a larger area for the list of time zones.</p>

## Date and Time Selector

### Date and Time Selection Functions

---

-> anyLocale      If true, ignore ioLocaleInTimeZoneP upon entry.

**Result**      Returns true if the user clicked the OK button in the dialog to change the time zone, or false if the user clicked the Cancel button.

**Comments**      The time zones displayed in the form are listed by country. For this reason, if the time zone specified by ioTimeZoneP is shared by several countries, you need to supply a value for ioLocaleTimeZoneP to identify which country should be selected when the list is first displayed. You can use the constant lmAnyLanguage as the value for the language field of the structure pointed to by this parameter.

If you don't care which value is initially selected, pass true for the anyLocale parameter. In this case, the first country that matches the GMT offset given in ioTimeZoneP is selected.

You might want to use the current time zone stored in the system preferences as the initial value for the ioTimeZoneP parameter. To obtain this time zone, do the following:

```
Int16 timeZone =
    (Int16) PrefGetPreference(prefTimeZone);
CountryType timeZoneCountry = (CountryType)
    PrefGetPreference(prefTimeZoneCountry);
LmLocaleType timeZoneLocale;
Boolean change;

timeZoneLocale.country = timeZoneCountry;
timeZoneLocale.language = lmAnyLanguage;
change = SelectTimeZone(&timeZone,
    &timeZoneLocale, NULL, true, false);
```

**Compatibility**      Implemented if [4.0 New Feature Set](#) is present.

# Fields

---

This chapter provides the following information about field objects:

- [Field Data Structures](#)
- [Field Resources](#)
- [Field Functions](#)

The header file `Field.h` declares the API that this chapter describes. For more information on fields, see the section “[Fields](#)” in the *Palm OS Programmer’s Companion*, vol. I.

## Field Data Structures

### FieldAttrType

The `FieldAttrType` bit field defines the visible characteristics of the field. The functions [`FldGetAttributes`](#) and [`FldSetAttributes`](#) return and set these values. There are other functions that retrieve or set individual attributes defined here. Those functions are noted below.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `FieldAttrType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    UInt16 usable      :1;
    UInt16 visible     :1;
    UInt16 editable    :1;
    UInt16 singleLine  :1;
    UInt16 hasFocus    :1;
```

## Fields

### *Field Data Structures*

---

```
    UInt16 dynamicSize      :1;
    UInt16 insPtVisible    :1;
    UInt16 dirty            :1;
    UInt16 underlined       :2;
    UInt16 justification    :2;
    UInt16 autoShift        :1;
    UInt16 hasScrollBar     :1;
    UInt16 numeric          :1;
} FieldAttrType;
```

### Field Descriptions

usable	If not set, the field object is not considered part of the current interface of the application, and it doesn't appear on screen. The function <a href="#">FldSetUsable</a> sets this value, but it is better to use <a href="#">FrmShowObject</a> .
visible	Set or cleared internally when the field object is drawn with <a href="#">FldDrawField</a> or <a href="#">FrmShowObject</a> , and erased with <a href="#">FldEraseField</a> or <a href="#">FrmHideObject</a> .
editable	If not set, the field object doesn't accept Graffiti® input or editing commands and the insertion point cannot be positioned with the pen. The text can still be selected and copied.
singleLine	If set, the field is a single line of text high and the text does not wrap when it exceeds the width of the field. If not set, the text wraps to fill multiple lines.
hasFocus	Set internally when the field has the current focus. The blinking insertion point appears in the field that has the current focus. Use the function <a href="#">FrmSetFocus</a> and <a href="#">FldReleaseFocus</a> to set this value.

dynamicSize	If set, a <code>fldHeightChangedEvent</code> is generated whenever the number of lines needs to increase or decrease. Your application needs to respond to this event by adjusting the size of the field's bounding box. If not set, the text wraps to fill more (or less) lines as required, but the event is not generated. <i>Note that this bit does not cause the field to change size automatically;</i> your application must respond to the <code>fldHeightChangedEvent</code> and resize the field itself.
insPtVisible	Set this attribute to <code>false</code> if the Single Line attribute is set.
dirty	If set, the insertion point is scrolled into view. This attribute is set and cleared internally.
underlined	If set, the user has modified the field. The functions <a href="#"><u>FldDirty</u></a> and <a href="#"><u>FldSetDirty</u></a> retrieve this field's value.
	If set each line of the field, including blank lines, is underlined. Possible values are defined by the <code>UnderlineModeType</code> defined in <code>Window.h</code> :
	<code>noUnderline</code> <code>grayUnderline</code> <code>solidUnderline</code> <code>colorUnderline</code>
	Editable text fields generally use <code>grayUnderline</code> as the value.
	The <code>solidUnderline</code> value is only valid for Palm OS 3.1 and higher.
	The <code>colorUnderline</code> value is only valid for Palm OS 3.5 and higher.

## Fields

### *Field Data Structures*

---

justification	Specifies the text alignment. Possible values are <code>leftAlign</code> and <code>rightAlign</code> . (left or right justification only; <code>centerAlign</code> justification is not supported).
autoShift	If set, Graffiti auto-shift rules are applied.
hasScrollBar	If set, the field has a scrollbar. The system sends more frequent <code>fldChangedEvent</code> s so the application can adjust the height appropriately.
numeric	If set, only the characters 0 through 9 and associated separators are allowed to be entered in the field. The associated separators are the thousands separator and the decimal character. The values of these two characters depend on the settings in the Formats prefs panel.

## FieldPtr

The `FieldPtr` type defines a pointer to a [FieldType](#) structure.

```
typedef FieldType *FieldPtr;
```

You pass the `FieldPtr` as an argument to all field functions. You can obtain the `FieldPtr` using the function [FrmGetObjectPtr](#) in this way:

```
fldPtr = FrmGetObjectPtr(frm,  
    FrmGetObjectIndex(frm, fldID));
```

where `fldID` is the resource ID assigned when you created the field.

## FieldType

The `FieldType` structure represents a field.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `FieldType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    UInt16           id;
    RectangleType    rect;
    FieldAttrType   attr;
    Char             *text;
    MemHandle        textHandle;
    LineInfoPtr      lines;
    UInt16           textLen;
    UInt16           textBlockSize;
    UInt16           maxChars;
    UInt16           selfFirstPos;
    UInt16           selLastPos;
    UInt16           insPtXPos;
    UInt16           insPtYPos;
    FontID           fontID;
    UInt8            reserved;
} FieldType;
```

Your code should treat the `FieldType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

## Field Descriptions

<code>id</code>	ID value you specified when you created the field resource. This ID value is included as part of the event data of <a href="#">fldEnterEvent</a> .
<code>rect</code>	Position and size of the field object. The functions <a href="#">FldGetBounds</a> , <a href="#">FrmGetObjectBounds</a> , <a href="#">FldSetBounds</a> , and <a href="#">FrmSetObjectBounds</a> retrieve and set this value.

## Fields

### *Field Data Structures*

---

attr	Field object attributes. (See <a href="#">FieldAttrType</a> .)
text	Pointer to the null-terminated string that is displayed by the field object. The functions <a href="#">FldGetTextPtr</a> and <a href="#">FldSetTextPtr</a> retrieve and set this value (see below).  Never set the contents of this string directly; for example, do not pass this pointer as the destination value to a function such as <code>StrCopy</code> .
textHandle	Handle to the stored text or to a database record containing the stored text. The functions <a href="#">FldGetTextHandle</a> and <a href="#">FldSetTextHandle</a> retrieve and set this value.  If <code>textHandle</code> is defined, the field calculates the <code>text</code> pointer when it locks the handle. In general, you should only use <a href="#">FldGetTextPtr</a> and <a href="#">FldSetTextPtr</a> on text fields that aren't editable. On editable text fields, use <a href="#">FldGetTextHandle</a> and <a href="#">FldSetTextHandle</a> .  Also note that editable text fields allocate the text handle as necessary. If a user starts typing in a field that doesn't have a text handle allocated, the field will allocate one. The field also resizes the text's memory block as necessary when the user adds more text.
lines	Pointer to an array of <code>LineInfoType</code> structures. There is one entry in this array for each visible line of the text. (See <a href="#">LineInfoType</a> .) The field code maintains this array internally; you should never change the <code>lines</code> array yourself.  Note that this value is NULL for single line fields, and for fields that do not have an allocated text handle.

<code>textLen</code>	Length in bytes of the string currently displayed by the field object; the null terminator is excluded. You can retrieve this value with <a href="#">FldGetTextLength</a> .
<code>textBlockSize</code>	Allocated size of the memory block that holds the field object's text string. You can retrieve this value with <a href="#">FldGetTextAllocatedSize</a> .  Fields allocate memory for the field text as needed, several bytes at a time.  Note that <code>textBlockSize</code> may be different from the size of the chunk pointed to by <code>textHandle</code> . The <code>textHandle</code> may point to a database record that contains, in part, the text displayed by the field. If you called <a href="#">MemHandleSize</a> on such a <code>textHandle</code> , the number returned may be greater than <code>textBlockSize</code> .
<code>maxChars</code>	Maximum number of bytes the field object accepts. The functions <a href="#">FldGetMaxChars</a> and <a href="#">FldSetMaxChars</a> retrieve and set this value.  Note the difference between <code>textLen</code> , <code>textBlockSize</code> , and <code>maxChars</code> . <code>textLen</code> is the number of bytes of character data that <code>text</code> actually holds. <code>textBlockSize</code> is the amount of memory currently allocated for the text (which must be greater than or equal to <code>textLen</code> ), and <code>maxChars</code> sets the maximum value that <code>textBlockSize</code> and <code>textLen</code> can expand to.  For example, if you've created a text field for users to enter their first names in, you might specify that the maximum length of this field is 20 bytes. If a user enters "John" in this field, <code>textLen</code> is 4, <code>textBlockSize</code> is 16, and <code>maxChars</code> is 20.

## Fields

### Field Data Structures

---

selFirstPos	Starting character offset in bytes of the current selection. Use <a href="#">FldGetSelection</a> and <a href="#">FldSetSelection</a> to retrieve and set this value and the selLastPos value.
selLastPos	Ending character offset in bytes of the current selection. When selFirstPos equals selLastPos, there is no selection.
insPtXPos	Horizontal location of the insertion point, given as the offset in bytes into the line indicated by insPtYPos. The functions <a href="#">FldGetInsPtPosition</a> and <a href="#">FldSetInsPtPosition</a> retrieve and set a byte offset calculated from this value. If the insertion point isn't visible—if it's on a line that's either above or below the set of visible lines—insPtXPos is the absolute byte offset of the insertion point.
insPtYPos	Vertical location of the insertion point, given as the display line where the insertion point is positioned. The first display line is zero. The first display line may be different from the first line of text in the field if the field has been scrolled. The functions <a href="#">FldGetInsPtPosition</a> and <a href="#">FldSetInsPtPosition</a> retrieve and set a byte offset calculated from this value. If the insertion point isn't visible—if it's on a line that's either above or below the set of visible lines—insPtYPos is set to 0x8000.
fontID	Font ID for the field. See <code>Font.h</code> for more information. The functions <a href="#">FldGetFont</a> and <a href="#">FldSetFont</a> retrieve and set this value.
reserved	Reserved for future use.

## LineInfoPtr

The `LineInfoPtr` type defines a pointer to the [LineInfoType](#).

```
typedef LineInfoType *LineInfoPtr;
```

## LineInfoType

The `LineInfoType` structure defines an element in the field's `lines` array. The `lines` array contains the field's word wrapping information. There is one element in the array per visible line in the field, including visible lines that contain no text. The field code maintains this array internally; you should never change the `lines` array yourself.

The functions `FldCalcFieldHeight`, `FldGetVisibleLines`, `FldRecalculateField`, and `FldGetNumberOfBlankLines` retrieve or set information in this structure. The scrolling functions `FldGetScrollPosition`, `FldGetScrollValues`, `FldScrollField`, and `FldSetScrollPosition` also retrieve or set information in this structure.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `LineInfoType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    UInt16      start;
    UInt16      length;
} LineInfoType;
```

## Field Descriptions

`start` The byte offset into the `FieldType`'s `text` field of the first character displayed by this line. If the line is blank, `start` is equal to `textLen` and `length` is 0.

`length` The length in bytes of the portion of the string displayed on this line. If the line is blank, the length is 0.

## Fields

### *Field Resources*

---

## Field Resources

The [Field Resource](#) (tFLD) represents a field on screen.

## Field Functions

### FldCalcFieldHeight

**Purpose** Determine the height of a field for a string.

**Declared In** Field.h

**Prototype** `UInt16 FldCalcFieldHeight (const Char *chars,  
                  UInt16 maxWidth)`

**Parameters** `-> chars` Pointer to a null-terminated string.  
`-> maxWidth` Maximum line width in pixels.

**Result** Returns the total number of lines needed to draw the string passed.

**Comments** The width of a field is contained in the `rect` member of the [FieldType](#) structure. You can retrieve this value in the following way:

```
FrmGetObjectBounds(frm,  
                  FrmGetObjectIndex(frm, fldID),  
                  &myRect);  
fieldWidth = myRect.extent.x;  
FldCalcFieldHeight(myString, fieldWidth);
```

**See Also** [FldWordWrap](#)

## FldCompactText

<b>Purpose</b>	Compact the memory block that contains the field's text to release any unused space.
<b>Declared In</b>	Field.h
<b>Prototype</b>	<code>void FldCompactText (FieldType *fldP)</code>
<b>Parameters</b>	<code>-&gt; fldP</code> Pointer to a field object ( <a href="#">FieldType</a> structure).
<b>Result</b>	Returns nothing.
<b>Comments</b>	<p>As characters are added to the field's text, the block that contains the text is grown. The block is expanded several bytes at a time so that it doesn't have to expand each time a character is added. This expansion may result in some unused space in the text block.</p> <p>Applications should call this function on field objects that edit data records in place before the field is unlocked, or at any other time when a compact field is desirable; for example, before writing to the storage heap.</p>
<b>See Also</b>	<a href="#">FldGetTextAllocatedSize</a> , <a href="#">FldSetTextAllocatedSize</a>

## FldCopy

<b>Purpose</b>	Copy the current selection to the text clipboard.
<b>Declared In</b>	Field.h
<b>Prototype</b>	<code>void FldCopy (const FieldType *fldP)</code>
<b>Parameters</b>	<code>-&gt; fldP</code> Pointer to a field object ( <a href="#">FieldType</a> structure).
<b>Result</b>	Returns nothing.
<b>Comments</b>	This function leaves the current selection highlighted.

## Fields

### *Field Functions*

---

This function replaces anything previously in the text clipboard if there is text to copy. If no text is selected, the function beeps and the clipboard remains intact.

**See Also** [FldCut](#), [FldPaste](#)

## FldCut

**Purpose** Copy the current selection to the text clipboard, delete the selection from the field, and redraw the field.

**Declared In** Field.h

**Prototype** void FldCut (FieldType \*fldP)

**Parameters** -> fldP      Pointer to a field object ([FieldType](#) structure).

**Result** Returns nothing.

**Comments** If text is selected, the text is removed from the field, the field's dirty attribute is set, and anything previously in the text clipboard is replaced by the selected text.

If there is no selection or if the field is not editable, this function beeps.

**See Also** [FldCopy](#), [FldPaste](#), [FldUndo](#)

## FldDelete

<b>Purpose</b>	Delete the specified range of characters from the field and redraw the field.						
<b>Declared In</b>	Field.h						
<b>Prototype</b>	<pre>void FldDelete (FieldType *fldP, UInt16 start,                 UInt16 end)</pre>						
<b>Parameters</b>	<table><tr><td>-&gt; fldP</td><td>Pointer to the field object (<a href="#">FieldType</a> structure) to delete from.</td></tr><tr><td>-&gt; start</td><td>The beginning of the range of characters to delete given as a valid byte offset into the field's <code>text</code> string.</td></tr><tr><td>-&gt; end</td><td>The end of the range of characters to delete given as a valid byte offset into the field's <code>text</code> string. On systems that support multi-byte characters, this position must be an inter-character boundary. That is, it must not point to a middle byte of a multi-byte character.</td></tr></table>	-> fldP	Pointer to the field object ( <a href="#">FieldType</a> structure) to delete from.	-> start	The beginning of the range of characters to delete given as a valid byte offset into the field's <code>text</code> string.	-> end	The end of the range of characters to delete given as a valid byte offset into the field's <code>text</code> string. On systems that support multi-byte characters, this position must be an inter-character boundary. That is, it must not point to a middle byte of a multi-byte character.
-> fldP	Pointer to the field object ( <a href="#">FieldType</a> structure) to delete from.						
-> start	The beginning of the range of characters to delete given as a valid byte offset into the field's <code>text</code> string.						
-> end	The end of the range of characters to delete given as a valid byte offset into the field's <code>text</code> string. On systems that support multi-byte characters, this position must be an inter-character boundary. That is, it must not point to a middle byte of a multi-byte character.						
<b>Result</b>	Returns nothing.						
<b>Comments</b>	<p>This function deletes all characters from the starting offset up to the ending offset and sets the field's dirty attribute. It does not delete the character at the ending offset.</p> <p>If <code>start</code> or <code>end</code> point to an intra-character boundary, <code>FldDelete</code> attempts to move the offset backward, toward the beginning of the text, until the offset points to an inter-character boundary (i.e., the start of a character).</p> <p><code>FldDelete</code> posts a <a href="#"><code>fldChangedEvent</code></a> to the event queue. If you call this function repeatedly, you may overflow the event queue with <code>fldChangedEvents</code>. An alternative is to remove the text</p>						

## Fields

### Field Functions

---

handle from the field, change the text, and then set the field's handle again. See [FldGetTextHandle](#) for a code example.

**See Also** [FldInsert](#), [FldEraseField](#), [TxtCharBounds](#)

## FldDirty

**Purpose** Return `true` if the field has been modified since the text value was set.

**Declared In** Field.h

**Prototype** Boolean FldDirty (const FieldType \*fldP)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).

**Result** Returns `true` if the field has been modified either by the user or through calls to certain functions such as [FldInsert](#) and [FldDelete](#), `false` if the field has not been modified.

**See Also** [FldSetDirty](#), [FieldAttrType](#)

## FldDrawField

**Purpose** Draw the text of the field.

**Declared In** Field.h

**Prototype** void FldDrawField (FieldType \*fldP)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).

**Result** Returns nothing.

**Comments** The field's `usable` attribute must be `true` or the field won't be drawn.

This function doesn't erase the area behind the field before drawing.

If the field has the focus, the blinking insertion point is displayed in the field.

**See Also** [FldEraseField](#)

## FldEraseField

**Purpose** Erase the text of a field and turn off the insertion point if it's in the field.

**Declared In** Field.h

**Prototype** void FldEraseField (FieldType \*fldP)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).

**Result** Returns nothing.

**Comments** You rarely need to call this function directly. Instead, use [FrmHideObject](#), which calls FldEraseField for you.

This function visibly erases the field from the display, but it doesn't modify the contents of the field or free the memory associated with it.

If the field has the focus, the blinking insertion point is turned off.

This function sets the `visible` attribute to `false`. (See [FieldAttrType](#).)

**See Also** [FldDrawField](#)

## Fields

### *Field Functions*

---

## FldFreeMemory

<b>Purpose</b>	Release the handle-based memory allocated to the field's text and the associated word-wrapping information.
<b>Declared In</b>	Field.h
<b>Prototype</b>	<code>void FldFreeMemory (FieldType *fldP)</code>
<b>Parameters</b>	<code>-&gt; fldP</code> Pointer to a field object ( <a href="#">FieldType</a> structure).
<b>Result</b>	Returns nothing. May raise a fatal error message if the text associated with the field is actually a record in a database.
<b>Comments</b>	<p>This function releases</p> <ul style="list-style-type: none"><li>• The memory allocated to the text of a field—the memory block that the <code>textHandle</code> member of the <code>FieldType</code> data structure points to. If the field's <code>textHandle</code> is <code>NULL</code> but there is a <code>text</code> string associated with that field (which is often the case with noneditable text fields), the text string is not freed.</li><li>• The memory allocated to hold the word-wrapping information—the memory block that the <code>lines</code> member of the <code>FieldType</code> data structure points to.</li></ul> <p>This function doesn't affect the display of the field. Fields allocate memory for the text string as needed, so it is not an error to call this function while the field is still displayed. That is, if <code>text</code> is <code>NULL</code> and the user starts typing in the field, the field simply allocates memory for text and continues.</p>

## FldGetAttributes

**Purpose** Return the attributes of a field.

**Declared In** Field.h

**Prototype** void FldGetAttributes (const FieldType \*fldP,  
FieldAttrPtr attrP)

**Parameters** -> fldP Pointer to a [FieldType](#) structure.  
-< attrP Pointer to the [FieldAttrType](#) structure.

**Result** Returns the field's attributes in the attrP parameter.

**See Also** [FldSetAttributes](#)

## FldGetBounds

**Purpose** Return the current bounds of a field.

**Declared In** Field.h

**Prototype** void FldGetBounds (const FieldType \*fldP,  
RectanglePtr rect)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).  
-< rect Pointer to a RectangleType structure.

**Result** Returns nothing. Stores the field's bounds in the RectangleType structure reference by rect.

**Comments** Returns the rect field of the FieldType structure.

**See Also** [FldSetBounds](#), [FrmGetObjectBounds](#)

## Fields

### *Field Functions*

---

## FldGetFont

**Purpose** Return the ID of the font used to draw the text of a field.

**Declared In** Field.h

**Prototype** FontID FldGetFont (const FieldType \*fldP)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).

**Result** Returns the ID of the font.

**See Also** [FldSetFont](#)

## FldGetInsPtPosition

**Purpose** Return the insertion point position within the string.

**Declared In** Field.h

**Prototype** UInt16 FldGetInsPtPosition  
(const FieldType \*fldP)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).

**Result** Returns the byte offset of the insertion point.

**Comments** The insertion point is to the left of the byte offset that this function returns. That is, if this function returns 0, the insertion point is to the left of the first character in the string. In multiline fields, line feeds are counted as a single character in the string, and the byte offset after the line feed character is the beginning of the next line.

**See Also** [FldSetInsPtPosition](#)

## FldGetMaxChars

**Purpose** Return the maximum number of bytes the field accepts.

**Declared In** Field.h

**Prototype** `UInt16 FldGetMaxChars (const FieldType *fldP)`

**Parameters** `-> fldP` Pointer to a field object ([FieldType](#) structure).

**Result** Returns the maximum length in bytes of characters the user is allowed to enter. This is the `maxChars` field in `FieldType`.

**See Also** [FldSetMaxChars](#)

## FldGetNumberOfBlankLines

**Purpose** Return the number of blank lines that are displayed at the bottom of a field.

**Declared In** Field.h

**Prototype** `UInt16 FldGetNumberOfBlankLines  
(const FieldType *fldP)`

**Parameters** `-> fldP` Pointer to a [FieldType](#) structure.

**Result** Returns the number of blank lines visible.

**Comments** This routine is useful for updating a scroll bar after characters have been removed from the text in a field. See the `NoteViewScroll` function in the Address sample application for an example.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## Fields

### Field Functions

---

## FldGetScrollPosition

<b>Purpose</b>	Return the offset of the first character in the first visible line of a field.
<b>Declared In</b>	Field.h
<b>Prototype</b>	<code>UInt16 FldGetScrollPosition (const FieldType *fldP)</code>
<b>Parameters</b>	<code>-&gt; fldP</code> Pointer to a field object ( <a href="#">FieldType</a> structure).
<b>Result</b>	Returns the offset of the first visible character.
<b>See Also</b>	<a href="#">FldSetScrollPosition</a> , <a href="#">LineInfoType</a>

## FldGetScrollValues

<b>Purpose</b>	Return the values necessary to update a scroll bar.
<b>Declared In</b>	Field.h
<b>Prototype</b>	<code>void FldGetScrollValues (const FieldType *fldP,                           UInt16 *scrollPosP, UInt16 *textHeightP,                           UInt16 *fieldHeightP)</code>
<b>Parameters</b>	<code>-&gt; fldP</code> Pointer to a <a href="#">FieldType</a> structure. <code>&lt;- scrollPosP</code> The line of text that is the topmost visible line. Line numbering starts with 0. <code>&lt;-textHeightP</code> The number of lines needed to display the field's text, given the width of the field. <code>&lt;-fieldHeightP</code> The number of visible lines in the field.
<b>Result</b>	Returns nothing. Stores the position, text height, and field height in the parameters passed in.

**Comments** Use the values returned by this function to calculate the values you send to [SclSetScrollBar](#) to update the scroll bar. For example:

```
FldGetScrollValues (fldP, &scrollPos,  
    &textHeight, &fieldHeight);  
  
if (textHeight > fieldHeight)  
    maxValue = textHeight - fieldHeight;  
else if (scrollPos)  
    maxValue = scrollPos;  
else  
    maxValue = 0;  
  
SclSetScrollBar (bar, scrollPos, 0, maxValue,  
    fieldHeight-1);  
}
```

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

**See Also** [FldSetCursorPosition](#)

## FldGetSelection

**Purpose** Return the current selection of a field.

**Declared In** Field.h

**Prototype** void FldGetSelection (const FieldType \*fldP,  
 UInt16 \*startPosition, UInt16 \*endPosition)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).  
-< startPosition Pointer to the start of the selected characters range, given as the byte offset into the field's text.

## Fields

### Field Functions

---

<- `endPosition`    Pointer to end of the selected characters range given as the byte offset into the field's text.

**Result**    Returns the starting and ending byte offsets in `startPosition` and `endPosition`.

**Comments**    The first character in a field is at offset zero.

If the user has selected the first five characters of a field, `startPosition` will contain the value 0 and `endPosition` the value 5, assuming all characters are a single byte long.

**See Also**    [FldSetSelection](#)

## FldGetTextAllocatedSize

**Purpose**    Return the number of bytes allocated to hold the field's text string. Don't confuse this number with the actual length of the text string displayed in the field.

**Declared In**    Field.h

**Prototype**    `UInt16 FldGetTextAllocatedSize  
(const FieldType *fldP)`

**Parameters**    `-> fldP`              Pointer to a field object.

**Result**    Returns the number of bytes allocated for the field's text. This is the `textBlockSize` field in [FieldType](#).

**See Also**    [FldSetTextAllocatedSize](#)

## FldGetTextHandle

<b>Purpose</b>	Return a handle to the block that contains the text string of a field.
<b>Declared In</b>	Field.h
<b>Prototype</b>	<code>MemHandle FldGetTextHandle (const FieldType *fldP)</code>
<b>Parameters</b>	<code>-&gt; fldP</code> Pointer to a field object ( <a href="#">FieldType</a> structure).
<b>Result</b>	Returns the handle to the text string of a field or NULL if no handle has been allocated for the field pointer.
<b>Comments</b>	The handle returned by this function is not necessarily the handle to the start of the string. If you've used <a href="#">FldSetText</a> to set the field's text to a string that is part of a database record, the text handle points to the start of that record. You'll need to compute the offset from the start of the record to the start of the string. You can either store the offset that you passed to <a href="#">FldSetText</a> or you can compute the offset by performing pointer arithmetic on the pointer you get by locking this handle and the pointer returned by <a href="#">FldGetTextPtr</a> .  If you are obtaining the text handle so that you can edit the field's text, you must remove the handle from the field before you do so. If you change the text while it is being used by a field, the field's internal structures specifying the text length, allocated size, and word wrapping information can become out of sync. To avoid this problem, remove the text handle from the field, change the text, and then set the field's text handle again. For example:
	<pre>/* Get the handle for the string and unlock */ /* it by removing it from the field. */ textH = FldGetTextHandle(fldP); FldSetTextHandle (fldP, NULL);  /* Insert code that modifies the string here.*/ /* The basic steps are: */ /* resize the chunk if necessary, */</pre>

## Fields

### Field Functions

---

```
/* lock the chunk, write to it, and then */
/* unlock the chunk. If the text is in a */
/* database record, use Data Manager calls. */

/* Update the text in the field. */
FldSetTextHandle (fldP, textH);
FldDrawField(fldP);
```

**See Also** [FldSetTextHandle](#), [FldGetTextPtr](#)

## FldGetTextHeight

**Purpose** Return the height in pixels of the number of visible lines that are not empty.

**Declared In** Field.h

**Prototype** UInt16 FldGetTextHeight (const FieldType \*fldP)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).

**Result** Returns the height in pixels of the number of visible lines that are not empty.

**Comments** Empty lines are all of the lines in the field following the last byte of text. Note that lines that contain only a linefeed are not empty. Also note that only lines that are visible are counted.

**See Also** [FldCalcFieldHeight](#)

## FldGetTextLength

**Purpose** Return the length in bytes of the field's text.

**Declared In** Field.h

**Prototype** UInt16 FldGetTextLength (const FieldType \*fldP)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).

**Result** Returns the length in bytes of a field's text, not including the terminating null character. This is the `textLen` field of `FieldType`.

## FldGetTextPtr

**Purpose** Return a locked pointer to the field's text string.

**Declared In** Field.h

**Prototype** Char \*FldGetTextPtr (const FieldType \*fldP)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).

**Result** Returns a locked pointer to the field's text string or NULL if the field is empty.

**Comments** The pointer returned by this function can become invalid if the user edits the text after you obtain the pointer.

Do not modify the contents of the pointer yourself. If you change the text while it is being used by a field, the field's internal structures specifying the text length, allocated size, and word wrapping information can become out of sync. To avoid this problem, follow the instructions given under [FldGetTextHandle](#).

## Fields

### Field Functions

---

**WARNING!** The pointer returned by this function is “owned” by the field until you specify a different pointer for the field. You should not store this pointer for future use, since the field can modify the size of the string, which can cause the pointer to become invalid.

---

**See Also** [FldSetTextPtr](#), [FldGetTextHandle](#)

## FldGetVisibleLines

**Purpose** Return the number of lines that can be displayed within the visible bounds of the field, regardless of what text is stored in the field.

**Declared In** Field.h

**Prototype** UInt16 FldGetVisibleLines (const FieldType \*fldP)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).

**Result** Returns the number of lines the field displays. (This is the size of the lines array in the [FieldType](#) structure.)

**See Also** [FldGetNumberOfBlankLines](#), [FldCalcFieldHeight](#)

## FldGrabFocus

**Purpose** Turn the insertion point on (if the specified field is visible) and position the blinking insertion point in the field.

**Declared In** Field.h

**Prototype** void FldGrabFocus (FieldType \*fldP)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).

**Result** Returns nothing.

**Comments** You rarely need to call this function directly. Instead, use [FrmSetFocus](#), which calls FldGrabFocus for you.

One instance where you need to call FldGrabFocus directly is to programmatically set the focus in a field that is contained in a table cell.

This function sets the field attribute hasFocus to true. (See [FieldAttrType](#).)

**See Also** [FrmSetFocus](#), [FldReleaseFocus](#)

## FldHandleEvent

**Purpose** Handles events that affect the field, including the following: [keyDownEvent](#), [penDownEvent](#), and [fldEnterEvent](#).

**Declared In** Field.h

**Prototype** Boolean FldHandleEvent (FieldType \*fldP,  
EventType \*eventP)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).  
-> eventP Pointer to an event (EventType data structure).

**Result** Returns true if the event was handled.

**Comments** When a [keyDownEvent](#) occurs in an editable text field, the keystroke appears in the field if it's a printable character or manipulates the insertion point if it's a "movement" character. The field is automatically updated.

When a [penDownEvent](#) occurs, the field sends a [fldEnterEvent](#) to the event queue.

When a [fldEnterEvent](#) occurs, the field grabs the focus. If the user has tapped twice in the current location, the word at that location is selected. If the user has tapped three times, the entire line is selected. Otherwise, the insertion point is placed in the specified position.

## Fields

### Field Functions

---

When a [menuCmdBarOpenEvent](#) occurs, the field adds paste, copy, cut, and undo buttons to the command toolbar. These buttons are only added if they make sense in the current context. That is, the cut button is only added if the field is editable, the paste button is only added if there is text on the clipboard and the field is editable, and the undo button is only added if there is an action to undo.

If the event alters the contents of the field, this function visually updates the field.

This function doesn't handle any events if the field is not editable or usable.

#### Compatibility

Double-tapping to select a word and triple-tapping to select a line are only supported if [3.5 New Feature Set](#) is present.

`FldHandleEvent` only handles the `menuCmdBarOpenEvent` if [3.5 New Feature Set](#) is present.

## FldInsert

**Purpose** Replace the current selection if any with the specified string and redraw the field.

**Declared In** `Field.h`

**Prototype** `Boolean FldInsert (FieldType *fldP,  
const Char *insertChars, UInt16 insertLen)`

**Parameters**

-> <code>fldP</code>	Pointer to the field object ( <a href="#">FieldType</a> structure) to insert to.
-> <code>insertChars</code>	Text string to be inserted.
-> <code>insertLen</code>	Length in bytes of the text string to be inserted, not counting the trailing null character.

**Result** Returns `true` if string was successfully inserted. Returns `false` if:

- The `insertLen` parameter is 0.
- The field is not editable.

- Adding the text would exceed the field's size limit (the `maxChars` value).
- More memory must be allocated for the field, and the allocation fails.

**Comments** If there is no current selection, the string passed is inserted at the position of the insertion point.

This function sets the field's dirty attribute and posts a [`fldChangedEvent`](#) to the event queue. If you call this function repeatedly, you may overflow the event queue with `fldChangedEvents`. An alternative is to remove the text handle from the field, change the text, and then set the field's handle again. See [`FldGetTextHandle`](#) for a code example.

**See Also** [`FldPaste`](#), [`FldDelete`](#), [`FldCut`](#), [`FldCopy`](#)

## **FldMakeFullyVisible**

**Purpose** Generates an event to cause a dynamically resizable field to expand its height to make its text fully visible.

**Declared In** `Field.h`

**Prototype** Boolean `FldMakeFullyVisible (FieldType *fldP)`

**Parameters** `-> fldP` Pointer to a field object ([`FieldType`](#) structure).

**Result** Returns `true` if the field is dynamically resizable and was not fully visible; `false` otherwise.

**Comments** Use this function on a field whose `dynamicSize` attribute is `true` (see [`FieldAttrType`](#)).

This function does not actually resize the field. Instead, it computes how big the field should be to be fully visible and then posts this information to the event queue in a [`fldHeightChangedEvent`](#).

## Fields

### Field Functions

---

---

**NOTE:** The event does not get generated if the number of lines in the field is equal to or greater than the value of the maximum lines attribute for the field.

---

If the field is contained in a table, the table's code handles the `fldHeightChangedEvent`. If the field is directly on a form, your application code should handle the `fldHeightChangedEvent` itself. The form code does not handle the event for you. Note that the constant `maxFieldLines` defines the maximum number of lines a field can expand to if the field is using the standard font.

**See Also** [TblHandleEvent](#)

## FldNewField

**Purpose** Create a new field object dynamically and install it in the specified form.

**Declared In** Field.h

**Prototype** `FieldType *FldNewField (void **formPP, UInt16 id, Coord x, Coord y, Coord width, Coord height, FontID font, UInt32 maxChars, Boolean editable, Boolean underlined, Boolean singleLine, Boolean dynamicSize, JustificationType justification, Boolean autoShift, Boolean hasScrollBar, Boolean numeric)`

**Parameters** `<-> formPP` Pointer to the pointer to the form in which the new field is installed. This value is not a handle; that is, the old form pointer value is not necessarily valid after this function returns. In subsequent calls, always use the new form pointer value returned by this function.

-> id	Symbolic ID of the field, specified by the developer. By convention, this ID should match the resource ID (not mandatory).
-> x	Horizontal coordinate of the upper-left corner of the field's boundaries, relative to the window in which it appears.
-> y	Vertical coordinate of the upper-left corner of the field's boundaries, relative to the window in which it appears.
-> width	Width of the field, expressed in pixels.
-> height	Height of the field, expressed in pixels.
-> font	Font to use to draw the field's text.
-> maxChars	Maximum number of bytes held by the field this function creates.
-> editable	Pass <code>true</code> to create a field in which the user can edit text. Pass <code>false</code> to create a field that cannot be edited.
-> underlined	Pass <code>noUnderline</code> for no underline, or <code>grayUnderline</code> to have the field underline the text it displays. On Palm OS® version 3.1 and higher, pass <code>solidUnderline</code> to use a solid underline instead of a dotted underline.
-> singleLine	Pass <code>true</code> to create a field that can display only a single line of text.
-> dynamicSize	Pass <code>true</code> to create a field that resizes dynamically according to the amount of text it displays.
-> justification	Pass either of the values <code>leftAlign</code> or <code>rightAlign</code> to specify left justification or right justification, respectively. The <code>centerAlign</code> value is not supported.
-> autoShift	Pass <code>true</code> to specify the use of Palm OS 2.0 (and later) auto-shift rules.

## Fields

### Field Functions

---

- > hasScrollBar Pass true to attach a scroll bar control to the field this function creates.
- > numeric Pass true to specify that only characters in the range of 0 through 9 are allowed in the field.

**Result** Returns a pointer to the new field object or NULL if there wasn't enough memory to create the field. Out of memory situations could be caused by memory fragmentation.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [FrmValidatePtr](#), [WinValidateHandle](#),  
[CtlValidatePointer](#), [FrmRemoveObject](#)

## FldPaste

**Purpose** Replace the current selection in the field, if any, with the contents of the text clipboard.

**Declared In** Field.h

**Prototype** void FldPaste (FieldType \*fldP)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).

**Result** Returns nothing

**Comments** The function performs these actions:

- Scrolls the field, if necessary, so the insertion point is visible.
- Inserts the clipboard text at the position of the insertion point if there is no current selection.
- Positions the insertion point after the last character inserted.
- Doesn't delete the current selection if there is no text in the clipboard.

**See Also** [FldInsert](#), [FldDelete](#), [FldCut](#), [FldCopy](#) [FldUndo](#)

## FldRecalculateField

<b>Purpose</b>	Update the structure that contains the word-wrapping information for each visible line.				
<b>Declared In</b>	Field.h				
<b>Prototype</b>	<pre>void FldRecalculateField (FieldType *fldP, Boolean redraw)</pre>				
<b>Parameters</b>	<table><tr><td>-&gt; fldP</td><td>Pointer to a field object (<a href="#">FieldType</a> structure).</td></tr><tr><td>-&gt; redraw</td><td>If true, redraws the field.</td></tr></table>	-> fldP	Pointer to a field object ( <a href="#">FieldType</a> structure).	-> redraw	If true, redraws the field.
-> fldP	Pointer to a field object ( <a href="#">FieldType</a> structure).				
-> redraw	If true, redraws the field.				
<b>Result</b>	Returns nothing.				
<b>Comments</b>	<p>This function will allocate the memory block that contains the displayed lines information if, and only if, the block does not yet exist.</p> <p>You should call this function when you change the field width or text length of the field. Do not call this function after changing the font or field height.</p> <p>Note that many of the field functions, including <a href="#">FldSetTextHandle</a>, <a href="#">FldInsert</a>, and <a href="#">FldDelete</a>, recalculate the word-wrapping information for you.</p>				
<b>Compatibility</b>	In releases prior to Palm OS 4.0, the word-wrapping information is only updated if the <code>redraw</code> parameter is set to true. As of Palm OS 4.0 it is updated whenever <code>FldRecalculateField</code> is called, regardless of the value of the <code>redraw</code> parameter.				

## Fields

### Field Functions

---

## FldReleaseFocus

**Purpose** Turn the blinking insertion point off if the field is visible and has the current focus, reset the Graffiti state, and reset the undo state.

**Declared In** Field.h

**Prototype** void FldReleaseFocus (FieldType \*fldP)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).

**Result** Returns nothing.

**Comments** This function sets the field attribute `hasFocus` to `false`. (See [FieldAttrType](#).)

Usually, you don't need to call this function. If the field is in a form or in a table that doesn't use custom drawing functions, the field code releases the focus for you when the focus changes to some other control. If your field is in any other type of object, such as a table that uses custom drawing functions or a gadget, you must call `FldReleaseFocus` when the focus moves away from the field.

**See Also** [FldGrabFocus](#)

## FldScrollable

**Purpose** Return `true` if the field is scrollable in the specified direction.

**Declared In** Field.h

**Prototype** Boolean FldScrollable (const FieldType \*fldP,  
WinDirectionType direction)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).

-> direction      The direction to test. `DirectionType` is defined in `Window.h`. It is an enum defining the constants up and down.

**Result**      Returns `true` if the field is scrollable in the specified direction; `false` otherwise.

**See Also**      [FldScrollField](#)

## FldScrollField

**Purpose**      Scroll a field up or down by the number of lines specified.

**Declared In**      `Field.h`

**Prototype**      `void FldScrollField (FieldType *fldP,  
                  UInt16 linesToScroll, WinDirectionType direction)`

**Parameters**

-> fldP	Pointer to a field object ( <a href="#">FieldType</a> structure).
-> linesToScroll	Number of lines to scroll.
-> direction	The direction to scroll. <code>DirectionType</code> is defined in <code>Window.h</code> . It is an enum defining, among others, the constants <code>winUp</code> and <code>winDown</code> .

**Result**      Returns nothing.

**Comments**      This function can't scroll horizontally, that is, right or left.

The field object is redrawn if it's scrolled; however, the scrollbar is not updated. Use [SclSetScrollBar](#) to update the scrollbar. For example:

```
FldScrollField (fldP, linesToScroll,
                  direction);

// Update the scroll bar.
SclGetScrollBar (bar, &value, &min, &max,
```

## Fields

### Field Functions

---

```
&pageSize) ;

if (direction == winUp)
    value -= linesToScroll;
else
    value += linesToScroll;

SclSetScrollBar (bar, value, min, max,
    pageSize);
```

If the field is not scrollable in the direction indicated, this function returns without performing any work. You can use [FldScrollable](#) before calling this function to see if the field can be scrolled.

**See Also** [FldScrollable](#), [FldSetScrollPosition](#)

## FldSendChangeNotification

**Purpose** Send a [fldChangedEvent](#) to the event queue.

**Declared In** Field.h

**Prototype** void FldSendChangeNotification  
(const FieldType \*fldP)

**Parameters** -> fldP Pointer to a field object.

**Result** Returns nothing.

**Comments** This function is used internally by the field code. You normally never call it in application code.

## FldSendHeightChangeNotification

**Purpose** Send a [fldHeightChangedEvent](#) to the event queue.

**Declared In** Field.h

**Prototype** void FldSendHeightChangeNotification  
(const FieldType \*fldP, UInt16 pos,  
Int16 numLines)

**Parameters** -> fldP Pointer to a field object.  
-> pos Character position of the insertion point.  
-> numLines New number of lines in the field.

**Result** Returns nothing.

**Comments** This function is used internally by the field code. You normally never call it in application code.

## FldSetAttributes

**Purpose** Set the attributes of a field.

**Declared In** Field.h

**Prototype** void FldSetAttributes (FieldType \*fldP,  
const FieldAttrType \*attrP)

**Parameters** -> fldP Pointer to a [FieldType](#) structure.  
-> attrP Pointer to the attributes.

**Result** Returns nothing.

**Comments** This function does not do anything to make the new attribute values take effect. For example, if you use this function to change the value of the underline attribute, you won't see its effect until you call [FldDrawField](#).

## Fields

### Field Functions

---

You usually do not have to modify field attributes at runtime, so you rarely need to call this function.

---

**WARNING!** You must not call this function to change any attributes that are noted as “for internal use only.”

---

The proper way to use `FldSetAttributes` is to:

1. Call `FldGetAttributes` to retrieve the attributes.
2. Set the specific flags that you want to modify.
3. Call `FldSetAttributes` to make the modifications.

---

**WARNING!** You must not call any field routines between calling `FldGetAttributes` and `FldSetAttributes`; this can cause the attributes to be out of sync, with unpredictable results.

---

**See Also** [FldGetAttributes](#), [FieldAttrType](#)

## FldSetBounds

**Purpose** Change the position or size of a field.

**Declared In** Field.h

**Prototype** void FldSetBounds (FieldType \*fldP,  
const RectangleType \*rP)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).  
-> rP Pointer to a [RectangleType](#) structure that contains the new bounds of the display.

**Result** Returns nothing. May raise a fatal error message if the memory block that contains the word-wrapping information needs to be resized and there is not enough space to do so.

**Comments** If the field is visible, the field is redrawn within its new bounds.

---

**NOTE:** You can change the height or location of the field while it's visible, but do not change the width.

---

## Fields

### Field Functions

---

The memory block that contains the word-wrapping information (see [LineInfoType](#)) will be resized if the number of visible lines is changed. The insertion point is assumed to be off when this routine is called.

Make sure that `rect` is at least as tall as a single line in the current font. (You can determine this value by calling [FntLineHeight](#).) If it's not, results are unpredictable.

**See Also** [FldGetBounds](#), [FrmSetObjectBounds](#)

## FldSetDirty

**Purpose** Set whether the field has been modified.

**Declared In** Field.h

**Prototype** void FldSetDirty (FieldType \*fldP, Boolean dirty)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).  
-> dirty true if the text is modified.

**Result** Returns nothing.

**Comments** You typically call this function when you want to clear the dirty attribute. The dirty attribute is set when the user enters or deletes text in the field. It is also set by certain field functions, such as [FldInsert](#) and [FldDelete](#).

**See Also** [FldDirty](#)

## FldSetFont

**Purpose** Set the font used by the field, update the word-wrapping information, and draw the field if the field is visible.

**Declared In** Field.h

**Prototype** void FldSetFont (FieldType \*fldP, FontID fontID)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).  
-> fontID ID of new font.

**Result** Returns nothing.

**See Also** [FldGetFont](#), [FieldAttrType](#)

## FldSetInsertionPoint

**Purpose** Set the location of the insertion point based on a specified string position.

**Declared In** Field.h

**Prototype** void FldSetInsertionPoint (FieldType \*fldP,  
UInt16 pos)

**Parameters** -> fldP Pointer to a [FieldType](#) structure.  
-> pos New location of the insertion point, given as a valid offset in bytes into the field's text. On systems that support multi-byte characters, you must make sure that this specifies an inter-character boundary (does not specify the middle or end bytes of a multi-byte character).

**Result** Nothing.

## Fields

### Field Functions

---

<b>Comments</b>	This routine differs from <a href="#">FldSetInsPtPosition</a> in that it doesn't make the character position visible. FldSetInsertionPoint also doesn't make the field the current focus of input if it was not already.
	If pos indicates a position beyond the end of the text in the field, the insertion point is set to the end of the field's text.
<b>Compatibility</b>	Implemented only if <a href="#">2.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TxtCharBounds</a>

## FldSetInsPtPosition

<b>Purpose</b>	Set the location of the insertion point for a given string position.				
<b>Declared In</b>	Field.h				
<b>Prototype</b>	<code>void FldSetInsPtPosition (FieldType *fldP, UInt16 pos)</code>				
<b>Parameters</b>	<table><tr><td>-&gt; fldP</td><td>Pointer to a field object (<a href="#">FieldType</a> structure).</td></tr><tr><td>-&gt; pos</td><td>New location of the insertion point, given as a valid offset in bytes into the field's text. On systems that support multi-byte characters, you must make sure that this specifies an inter-character boundary (does not specify the middle or end bytes of a multi-byte character).</td></tr></table>	-> fldP	Pointer to a field object ( <a href="#">FieldType</a> structure).	-> pos	New location of the insertion point, given as a valid offset in bytes into the field's text. On systems that support multi-byte characters, you must make sure that this specifies an inter-character boundary (does not specify the middle or end bytes of a multi-byte character).
-> fldP	Pointer to a field object ( <a href="#">FieldType</a> structure).				
-> pos	New location of the insertion point, given as a valid offset in bytes into the field's text. On systems that support multi-byte characters, you must make sure that this specifies an inter-character boundary (does not specify the middle or end bytes of a multi-byte character).				
<b>Result</b>	Returns nothing.				
<b>Comments</b>	If the position is beyond the visible text, the field is scrolled until the position is visible.				
<b>See Also</b>	<a href="#">FldGetInsPtPosition</a> , <a href="#">TxtCharBounds</a>				

## FldSetMaxChars

**Purpose** Set the maximum number of bytes the field accepts (the maxChars value).

**Declared In** Field.h

**Prototype** void FldSetMaxChars (FieldType \*fldP,  
                  UInt16 maxChars)

**Parameters** -> fldP                   Pointer to a field object ([FieldType](#) structure).  
                  -> maxChars               Maximum size in bytes of the characters the user may enter. You may specify any value up to maxFieldTextLen.

**Result** Returns nothing.

**Comments** Line feed characters are counted when the length of characters is determined.

**See Also** [FldGetMaxChars](#)

## FldSetMaxVisibleLines

**Purpose** Allows the creation of tables and fields smaller than 121 pixels tall that still drag-select when there are more lines of text than will fit in the space provided.

**Declared In** Field.h

**Prototype** void FldSetMaxVisibleLines (FieldType \*fldP,  
                  UInt8 maxLines)

**Parameters** -> fldP                   Pointer to a field object ([FieldType](#) structure).

## Fields

### *Field Functions*

---

-> maxLines      Maximum number of lines to which the field will visually grow.

**Result**      Returns nothing.

**Comments**      A field can be dynamically expandable. When it is, the field package needs to know the maximum number of lines that should be visible so it can prevent the field from being expanded further. Since field expansion is actually handled by enclosing objects—tables or forms—this function's primary purpose is to allow the enclosing object to tell the field how big it can get.

By default, tables assume that the field can get as big as the table.

If you don't call this function, fields expect to be at least 121 pixels tall and try to grow repeatedly until they are.

## FldSetScrollPosition

**Purpose**      Scroll the field such that the character at the indicated offset is the first character on the first visible line. Redraw the field if necessary.

**Declared In**      Field.h

**Prototype**      void FldSetScrollPosition (FieldType \*fldP,  
                          UInt16 pos)

**Parameters**      -> fldP      Pointer to a field object ([FieldType](#) structure).  
                          -> pos      Byte offset into the field's text string of first character to be made visible. On systems that support multi-byte characters, you must make sure that this specifies an inter-character boundary (does not specify the middle or end bytes of a multi-byte character).

**Result**      Returns nothing.

**Comments**      This function scrolls the field but does not update the field's scrollbar. You should update the scrollbar after calling this function.

To do so, first call [FldGetScrollValues](#) to determine the values to use, and then call [SclSetScrollBar](#).

**See Also** [FldGetPosition](#), [FldScrollField](#), [TxtCharBounds](#)

## FldSetSelection

**Purpose** Set the current selection in a field and highlight the selection if the field is visible.

**Declared In** Field.h

**Prototype** void FldSetSelection (FieldType \*fldP,  
                  UInt16 startPosition, UInt16 endPosition)

**Parameters**

-> fldP	Pointer to a field object ( <a href="#">FieldType</a> structure).
-> startPosition	Starting offset of the character range to highlight, given as a byte offset into the field's text.
-> endPosition	Ending offset of the character range to highlight. The ending offset should be greater than or equal to the starting offset. On systems that support multi-byte characters, this position must be an inter-character boundary. That is, it must not point to a middle byte of a multi-byte character.

**Result** Returns nothing.

**Comments** To cancel a selection, set both `startPosition` and `endPosition` to the same value. If `startPosition` equals `endPosition`, then the current selection is unhighlighted.

If either `startPosition` or `endPosition` point to an intra-character boundary, `FldSetSelection` attempts to move that

## Fields

### Field Functions

---

offset backward, toward the beginning of the string, until the offset points to an inter-character boundary (i.e., the start of a character).

**See Also** [TxtCharBounds](#)

## FldSetText

**Purpose** Set the text value of the field without updating the display.

**Declared In** Field.h

**Prototype** void FldSetText (FieldType \*fldP,  
MemHandle textHandle, UInt16 offset, UInt16 size)

**Parameters**

-> fldP	Pointer to a field object ( <a href="#">FieldType</a> structure).
-> textHandle	Unlocked handle of a block containing a null-terminated text string. Pass NULL for this parameter to remove the association between the field and the string it is currently displaying so that the string is not freed with the field when the form is deleted.
-> offset	Offset from start of block to start of the text string.
-> size	The allocated size of the text string. This is not the string length, and should not be set to 0, unless you are setting the text to the empty string.

**Result** Returns nothing.

**Comments** This function allows applications to perform editing in place in memory. You can use it to point the field to a string in a database record so that you can edit that string directly using field routines. As characters are added to the field's text, the block that contains the text is grown. So that the block doesn't have to be expanded for each character, it is expanded several bytes at a time; this expansion may result in some unused space in the text block. As characters are

removed from the field's text, the space is not automatically reclaimed. Because adding or removing characters when editing a data record in place may result in unused space at the end of the field's text block, applications should call [FldCompactText](#) on before the field is unlocked to release any unused space.

The handle that you pass to this function is assumed to contain a null-terminated string starting at `offset` bytes in the memory chunk. The string should be between 0 and `size - 1` bytes in length. The field does not make a copy of the memory chunk or the string data; instead, it stores the handle to the record in its structure.

---

**WARNING!** You cannot use this function to set two fields on a form so that they share a single string value. Thus, for instance, if you have a single string containing a person's name you cannot call `FldSetText` twice with the same string (but a different offset) to set up a first name field and a last name field.

---

`FldSetText` updates the word-wrapping information and places the insertion point after the last visible character, but it does not update the display. You must call [FldDrawField](#) after calling this function to update the display.

`FldSetText` increments the lock count for `textHandle` and decrements the lock count of its previous text handle (if any).

Because `FldSetText` (and `FldSetTextHandle`) may be used to edit database records, they do not free the memory associated with the previous text handle. If the previous text handle points to a string on the dynamic heap and you want to free it, use [FldGetTextHandle](#) to obtain the handle before using `FldSetText` and then free that handle after using `FldSetText`. (See [FldSetTextHandle](#) for a code example.)

If the field points to a database record, you want the memory associated with the text handle to persist; however, this memory and all other memory associated with the field is freed when the field itself is freed, which happens when the form is closed. If you don't want the memory associated with the text handle freed when the field is freed, use `FldSetText` and pass `NULL` for the text handle immediately before the form is closed. Passing `NULL`

## Fields

### *Field Functions*

---

removes the association between the field and the text handle that you want retained. That text handle is unlocked as a result of the `FldSetText` call, and when the field is freed, there is no text handle to free with it.

**See Also** [FldSetTextPtr](#), [FldSetTextHandle](#)

## **FldSetTextAllocatedSize**

**Purpose** Set the number of bytes allocated to hold the field's text string. Don't confuse this with the actual length of the text string displayed in the field.

**Declared In** `Field.h`

**Prototype** `void FldSetTextAllocatedSize (FieldType *fldP,  
                  UInt16 allocatedSize)`

**Parameters** `-> fldP` Pointer to a field object ([FieldType](#) structure).  
`-> allocatedSize` Number of bytes to allocate for the text.

**Result** Returns nothing.

**Comments** This function generally is not used. It does not resize the field's allocated memory for the text string; it merely sets the `textBlockSize` field of the `FieldType` structure. The value of this field is computed and maintained internally by the field, so you should not have to call `FldSetTextAllocatedSize` directly.

**See Also** [FldGetTextAllocatedSize](#), [FldCompactText](#)

## FldSetTextHandle

<b>Purpose</b>	Set the text value of a field to the string associated with the specified handle. Does not update the display.				
<b>Declared In</b>	Field.h				
<b>Prototype</b>	<pre>void FldSetTextHandle (FieldType *fldP, MemHandle textHandle)</pre>				
<b>Parameters</b>	<table><tr><td>-&gt; fldP</td><td>Pointer to a field object (<a href="#">FieldType</a> structure).</td></tr><tr><td>-&gt; textHandle</td><td>Unlocked handle of a field's text string. Pass NULL for this parameter to remove the association between the field and the string it is currently displaying so that the string is not freed with the field when the form is deleted.</td></tr></table>	-> fldP	Pointer to a field object ( <a href="#">FieldType</a> structure).	-> textHandle	Unlocked handle of a field's text string. Pass NULL for this parameter to remove the association between the field and the string it is currently displaying so that the string is not freed with the field when the form is deleted.
-> fldP	Pointer to a field object ( <a href="#">FieldType</a> structure).				
-> textHandle	Unlocked handle of a field's text string. Pass NULL for this parameter to remove the association between the field and the string it is currently displaying so that the string is not freed with the field when the form is deleted.				
<b>Result</b>	Returns nothing.				
<b>Comments</b>	<p>This function differs from <a href="#">FldSetText</a> in that it uses the entire memory chunk pointed to by <code>textHandle</code> for the string. In fact, this function simply calls <code>FldSetText</code> with an offset of 0 and a size equal to the entire length of the memory chunk. Use it to have the field edit a string in a database record if the entire record consists of that string, or use it to have the field edit a string in the dynamic heap.</p> <p>As characters are added to the field's text, the block that contains the text is grown. So that the block doesn't have to be expanded for each character, it is expanded several bytes at a time; this expansion may result in some unused space in the text block. As characters are removed from the field's text, the space is not automatically reclaimed. Because adding or removing characters when editing a data record in place may result in unused space at the end of the field's text block, applications should call <a href="#">FldCompactText</a> on before the field is unlocked to release any unused space.</p> <p><code>FldSetTextHandle</code> updates the word-wrapping information and places the insertion point after the last visible character, but it does</p>				

## Fields

### Field Functions

---

not update the display. You must call [FldDrawField](#) after calling this function to update the display.

`FldSetTextHandle` increments the lock count for `textHandle` and decrements the lock count of its previous text handle (if any).

Because `FldSetTextHandle` (and `FldSetText`) may be used to edit database records, they do not free the memory associated with the previous text handle. If the previous text handle points to a string on the dynamic heap and you want to free it, use [FldGetTextHandle](#) to obtain the handle before using `FldSetText` and then free that handle after using `FldSetText`. For example:

```
/* get the old text handle */
oldTxtH = FldGetTextHandle(fldP);

/* change the text and update the display */
FldSetTextHandle(fldP, txtH);
FldDrawField(fldP);

/* free the old text handle */
if (oldTxtH != NULL)
    MemHandleFree(oldTxtH);
```

If the field points to a database record, you want the memory associated with the text handle to persist; however, this memory and all other memory associated with the field is freed when the field itself is freed, which happens when the form is closed. If you don't want the memory associated with the text handle freed when the field is freed, use `FldSetTextHandle` and pass `NULL` for the text handle immediately before the form is closed. Passing `NULL` removes the association between the field and the text handle that you want retained. That text handle is unlocked as a result of the `FldSetTextHandle` call, and when the field is freed, there is no text handle to free with it.

**See Also** [FldSetTextPtr](#), [FldSetText](#)

## FldSetTextPtr

**Purpose** Set a noneditable field's text to point to the specified text string.

**Declared In** Field.h

**Prototype** void FldSetTextPtr (FieldType \*fldP, Char \*textP)

**Parameters** -> fldP Pointer to a field object ([FieldType](#) structure).  
-> textP Pointer to a null-terminated string.

**Result** Returns nothing. May display an error message if passed an editable text field.

**Comments** Do not call `FldSetTextPtr` with an editable text field. Instead, call [FldSetTextHandle](#) for editable text fields. `FldSetTextPtr` is intended for displaying noneditable text in the user interface.

If the field has more than one line, use [FldRecalculateField](#) to recalculate the word wrapping.

This function does **not** visually update the field. Use [FldDrawField](#) to do so.

The field never frees the string that you pass to this function, even when the field itself is freed. You must free the string yourself. Before you free the string, make sure the field is not still displaying it. Set the field's string pointer to some other string or call `FldSetTextPtr(fldP, NULL)` before freeing a string you have passed using this function.

**See Also** [FldSetTextHandle](#), [FldGetTextPtr](#)

## Fields

### *Field Functions*

---

## FldSetUsable

**Purpose** Set a field to usable or nonusable.

**Declared In** Field.h

**Prototype** void FldSetUsable (FieldType \*fldP,  
Boolean usable)

**Parameters** fldP Pointer to a [FieldType](#) structure.  
usable true to set usable; false to set nonusable.

**Result** Returns nothing.

**Comments** A nonusable field doesn't display or accept input.  
Use [FrmHideObject](#) and [FrmShowObject](#) instead of using this function.

**See Also** [FldEraseField](#), [FldDrawField](#), [FieldAttrType](#)

## FldUndo

**Purpose** Undo the last change made to the field object, if any. Changes include typing, backspaces, delete, paste, and cut.

**Declared In** Field.h

**Prototype** void FldUndo (FieldType \*fldP)

**Parameters** fldP Pointer to the field ([FieldType](#) structure) that has the focus.

**Result** Returns nothing.

**See Also** [FldPaste](#), [FldCut](#), [FldDelete](#), [FldInsert](#)

## FldWordWrap

**Purpose** Given a string and a width, return the number of bytes of characters that can be displayed using the current font.

**Declared In** Field.h

**Prototype** `UInt16 FldWordWrap (const Char *chars,  
Int16 maxWidth)`

**Parameters** `-> chars` Pointer to a null-terminated string.  
`-> maxWidth` Maximum line width in pixels.

**Result** Returns the length in bytes of the characters that can be displayed.

**See Also** [FntWordWrap](#)

## **Fields**

### *Field Functions*

---

# Find

---

This chapter describes the global find facility API declared in the header file `Find.h`.

## Find Functions

### FindDrawHeader

**Purpose** Draw the header line that separates, by application, the list of found items.

**Declared In** `Find.h`

**Prototype** Boolean `FindDrawHeader (FindParamsPtr findParams,  
Char const* title)`

**Parameters** `-> findParams` Pointer to the [sysAppLaunchCmdFind](#) launch code's parameter block.  
`-> title` String to display as the title for the current application.

**Result** Returns `true` if Find screen is filled up. Applications should exit from the search if this occurs.

**Comments** Call this function once at the beginning of your application's response to the [sysAppLaunchCmdFind](#) launch code. This function draws a header for your application's Find results. The header separates the search results from your application with the search results from another application.  
If your application searches multiple databases, you may also use `FindDrawHeader` as a separator between databases.

## Find

### *Find Functions*

---

## FindGetLineBounds

<b>Purpose</b>	Returns the bounds of the next available line for displaying a match in the Find results dialog.				
<b>Declared In</b>	Find.h				
<b>Prototype</b>	<pre>void FindGetLineBounds (const FindParamsType *findParams, RectanglePtr r)</pre>				
<b>Parameters</b>	<table><tr><td>-&gt; findParams</td><td>Pointer to the <a href="#">sysAppLaunchCmdFind</a> launch code's parameter block.</td></tr><tr><td>&lt;- r</td><td>The bounds of the area that should contain the next line of results.</td></tr></table>	-> findParams	Pointer to the <a href="#">sysAppLaunchCmdFind</a> launch code's parameter block.	<- r	The bounds of the area that should contain the next line of results.
-> findParams	Pointer to the <a href="#">sysAppLaunchCmdFind</a> launch code's parameter block.				
<- r	The bounds of the area that should contain the next line of results.				
<b>Result</b>	Returns nothing.				

## FindSaveMatch

<b>Purpose</b>	Saves the record and position within the record of a text search match. This information is saved so that it's possible to later navigate to the match.						
<b>Declared In</b>	Find.h						
<b>Prototype</b>	<pre>Boolean FindSaveMatch (FindParamsPtr findParams, UInt16 recordNum, UInt16 pos, UInt16 fieldNum, UInt32 appCustom, UInt16 cardNo, LocalID dbID)</pre>						
<b>Parameters</b>	<table><tr><td>-&gt; findParams</td><td>Pointer to the <a href="#">sysAppLaunchCmdFind</a> launch code's parameter block.</td></tr><tr><td>-&gt; recordNum</td><td>Record index. This parameter sets the recordNum field in the <a href="#">sysAppLaunchCmdGoto</a>'s parameter block.</td></tr><tr><td>-&gt; pos</td><td>Offset of the match string from start of record. This parameter sets the matchPos field in the <a href="#">sysAppLaunchCmdGoto</a>'s parameter block.</td></tr></table>	-> findParams	Pointer to the <a href="#">sysAppLaunchCmdFind</a> launch code's parameter block.	-> recordNum	Record index. This parameter sets the recordNum field in the <a href="#">sysAppLaunchCmdGoto</a> 's parameter block.	-> pos	Offset of the match string from start of record. This parameter sets the matchPos field in the <a href="#">sysAppLaunchCmdGoto</a> 's parameter block.
-> findParams	Pointer to the <a href="#">sysAppLaunchCmdFind</a> launch code's parameter block.						
-> recordNum	Record index. This parameter sets the recordNum field in the <a href="#">sysAppLaunchCmdGoto</a> 's parameter block.						
-> pos	Offset of the match string from start of record. This parameter sets the matchPos field in the <a href="#">sysAppLaunchCmdGoto</a> 's parameter block.						

-> fieldNum	Field number that the string was found in. This parameter sets the matchFieldNum field in the <a href="#">sysAppLaunchCmdGoto</a> 's parameter block.
-> appCustom	Extra data the application can save with a match. This parameter sets the matchCustom field in the <a href="#">sysAppLaunchCmdGoto</a> 's parameter block.
-> cardNo	Card number of the database that contains the match. This parameter sets the dbCardNo field in the <a href="#">sysAppLaunchCmdGoto</a> 's parameter block.
-> dbID	Local ID of the database that contains the match. This parameter sets the dbID field in the <a href="#">sysAppLaunchCmdGoto</a> 's parameter block.

**Result** Returns true if Find screen is filled up. Applications should exit from the search if this occurs.

**Comments** Call this function when your application finds a record with a matching string ([FindStrInStr](#) or [TxtFindString](#) returns true). This function saves the information you pass. If the user clicks this selection in the Find results dialog, the information is retrieved and used to set up the [sysAppLaunchCmdGoto](#) launch code's parameter block.

You can use the appCustom field for any application-specific data that might be needed to navigate to the record if the user selects it. It's common for localizable applications to set appCustom to the length of the matching string because the global find facility cannot correctly determine the length of the matching string on systems with multi-byte character sets. In some character encodings, one character may be accurately represented as either a single-byte character or a multi-byte character. The [TxtFindString](#) function accurately matches single-byte characters against their multi-byte equivalents and returns the length of the matching string. If you pass [TxtFindString](#)'s return value as the appCustom parameter to [FindSaveMatch](#), the matchCustom field of the

## Find

### *Find Functions*

---

sysAppLaunchCmdGoTo parameter block contains the length of the matching string.

If your application requires more custom information, you can store the information in a feature and store the feature number in the appCustom field. See the “[Feature Manager](#)” chapter for more information.

## FindStrInStr

**Purpose** Perform a case-blind prefix search for a string in another string. This function assumes that the string to find has already been normalized for searching.

**Declared In** Find.h

**Prototype** Boolean FindStrInStr (Char const \*strToSearch,  
Char const \*strToFind, UInt16 \*posP)

**Parameters**

-> strToSearch	String to search.
-> strToFind	Normalized version of the text string to be found.
<- posP	If a match is found, contains the offset of the match within strToSearch.

**Result** Returns true if the string was found. FindStrInStr matches the beginnings of words only; that is, strToFind must be a prefix of one of the words in strToSearch for FindStrInStr to return true.

**Comment** Don't use this function on systems that support the text manager. Instead, use [TxtFindString](#), which performs searches on strings that contain multi-byte characters and returns the length of the matching text.

For backward compatibility with systems that don't support the text manager, use [TxtGlueFindString](#), found in the PalmOSGlue library. [TxtGlueFindString](#) calls [TxtFindString](#) if the text

manager is present, or `FindStrInStr` if it is not present. For more information, see [Chapter 75, “PalmOSGlue Library”](#).

The method by which a search string is normalized varies depending on the version of Palm OS® and the character encoding supported by the device. The string passed to your application in the `strToFind` field of the [`sysAppLaunchCmdFind`](#) launch code parameter block has already been normalized. It can be passed directly to `FindStrInStr`, `TxtFindString`, or `TxtGlueFindString`. If you need to create your own normalized search string, use `TxtGluePrepFindString`, also in the PalmOSGlue library.

## **Find**

*Find Functions*

---

# Forms

---

This chapter provides the following information about form objects:

- [Form Data Structures](#)
- [Form Constants](#)
- [Form Resources](#)
- [Form Functions](#)
- [Application-Defined Functions](#)

The header file `Form.h` declares the API that this chapter describes. For more information on forms, see the section “[Text](#)” in the *Palm OS Programmer’s Companion*, vol. I.

## Form Data Structures

### FormAttrType

The `FormAttrType` bit field defines the visible characteristics of the form.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `FormAttrType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    UInt16 usable      :1;
    UInt16 enabled     :1;
    UInt16 visible     :1;
    UInt16 dirty       :1;
    UInt16 saveBehind  :1;
    UInt16 graffitiShift :1;
```

## Forms

### Form Data Structures

---

```
    UInt16 globalsAvailable : 1;  
    UInt16 doingDialog : 1;  
    UInt16 exitDialog : 1;  
    UInt16 reserved : 7;  
    UInt16 reserved2;  
} FormAttrType;
```

Your code should treat the FormAttrType bit field as opaque. Do not attempt to change bit field member values directly.

### Field Descriptions

usable	Not set if the form is not considered part of the current interface of the application, and it doesn't appear on screen.
enabled	Not used.
visible	Set or cleared internally when the field object is drawn or erased.
dirty	Not used.
saveBehind	Set if the bits behind the form are saved when the form is drawn.
graffitiShift	Set if the graffiti shift indicator is supported.
globalsAvailable	System use only.
doingDialog	System use only.
exitDialog	System use only.
reserved	Reserved for system use.
reserved2	Reserved for system use.

### Compatibility

The globalsAvailable, doingDialog, and exitDialog flags are present only if [3.5 New Feature Set](#) is present.

## FormBitmapType

The FormBitmapType structure defines the visible characteristics of a bitmap on a form.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the FormBitmapType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    FormObjAttrType attr;
    PointType pos;
    UInt16 rscID;
} FormBitmapType;
```

### Field Descriptions

attr	See <a href="#">FormObjAttrType</a> .
pos	Location of the bitmap.
rscID	Resource ID of the bitmap. If you use <a href="#">DmGetResource</a> with this value as the resource ID, it returns a pointer to a <a href="#">BitmapType</a> structure.

## FormFrameType

The FormFrameType structure defines a frame that appears on the form.

```
typedef struct {
    UInt16 id;
    FormObjAttrType attr;
    RectangleType rect;
    UInt16 frameType;
} FormFrameType;
```

## Forms

### *Form Data Structures*

---

#### Field Descriptions

id	ID of the frame.
attr	See <a href="#">FormObjAttrType</a> .
rect	Location and size of the frame.
frameType	The type of frame.

### FormGadgetAttrType

The FormGadgetAttrType bit field defines a gadget's attributes.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the FormGadgetAttrType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    UInt16 usable : 1;
    UInt16 extended : 1;
    UInt16 visible : 1;
    UInt16 reserved : 13;
} FormGadgetAttrType;
```

Your code should treat the FormGadgetAttrType structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

## Field Descriptions

- usable Not set if the gadget is not considered part of the current interface of the application, and it doesn't appear on screen. This is set by [FrmShowObject](#) and cleared by [FrmHideObject](#).
- extended If set, the gadget is an extended gadget. Extended gadgets are supported if [3.5 New Feature Set](#) is present. An extended gadget has the handler field defined in its [FormGadgetType](#). If not set, the gadget is a standard gadget compatible with all releases of Palm OS®.
- visible Set or cleared when the gadget is drawn or erased. [FrmHideObject](#) clears this value. You should set it explicitly in the gadget's callback function (if it has one) in response to a draw request.
- reserved Reserved for future use.

Many form functions ([FrmGetObject](#), [FrmHideObject](#), and [FrmGetObjectBounds](#), for example) take an object index as one of their arguments. The most common way to get an object's index is to call [FrmGetObjectIndex](#). [FrmGetObjectIndex](#) takes a form ID and returns the form object's index. This is the routine one should use in most cases, because the application usually knows the object ID. However, gadgets and specifically extended gadgets, have APIs with callbacks that pass back the gadget pointer and not the ID. In those cases, the only way to get the object index (so one can use the [FrmGetObject\\*](#) APIs) is to use the function [FrmGetObjectIndexFromPtr](#).

If you need the same functionality on pre-Palm OS 4.0 systems then you can accomplish the same thing with the following code snippet.

```
UInt16 index;
UInt16 objIndex = frmInvalidObjectId;
UInt16 numObjects = FrmGetNumberOfObjects(frmP)
for (index = 0; index < numObjects; index++) {
    if (FrmGetObjectPtr(index) == myObjPtr) {
        // Found it
        objIndex = index;
        break;
```

## Forms

### *Form Data Structures*

---

```
    }
```

---

**Compatibility** This type is defined only if [3.5 New Feature Set](#) is present.

## FormGadgetType

The FormGadgetType structure defines a gadget object that appears on a form.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the FormGadgetType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    UInt16 id;
    FormGadgetAttrType attr;
    RectangleType rect;
    const void *data;
    FormGadgetHandlerType *handler;
} FormGadgetType;
```

Your code should treat the FormGadgetType structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

### Field Descriptions

id	ID of the gadget resource.
attr	See <a href="#">FormGadgetAttrType</a> .
rect	Location and size of the object.
data	Pointer to any specific data that needs to be stored. You can set and retrieve the value of this field with <a href="#">FrmGetGadgetData</a> and <a href="#">FrmSetGadgetData</a> .
handler	Pointer to a callback function that controls the gadget's behavior and responds to events. You can set this field with <a href="#">FrmSetGadgetHandler</a> .

Many form functions ([FrmGetObjectType](#), [FrmHideObject](#), and [FrmGetObjectBounds](#), for example) take an object index as one of their arguments. The most common way to get an object's index is to

## Forms

### Form Data Structures

---

call [FrmGetObjectIndex](#). FrmGetObjectIndex takes a form ID and returns the form object's index. This is the routine one should use in most cases, because the application usually knows the object ID. However, gadgets have APIs with callbacks that pass back the gadget pointer and not the ID. In those cases, the only way to get the object index (so one can use the FrmGetObject\* APIs) is to use the function [FrmGetObjectIndexFromPtr](#).

If you need the same functionality on pre-Palm OS 4.0 systems then you can accomplish the same thing with the following code snippet.

---

```
UInt16 index;
UInt16 objIndex = frmInvalidObjectID;
UInt16 numObjects = FrmGetNumberOfObjects(frmP)
for (index = 0; index < numObjects; index++) {
    if (FrmGetObjectPtr(index) == myObjPtr) {
        // Found it
        objIndex = index;
        break;
    }
}
```

---

### Compatibility

In Palm OS® releases prior to 3.5, the attr field was of type [FormObjAttrType](#) and the handler field did not exist.

## FormGadgetTypeInCallback

The FormGadgetTypeInCallback structure is passed to your extended gadget handler and is identical to [FormGadgetType](#) except that its contents are *not* hidden when DO\_NOT\_ALLOW\_ACCESS\_TO\_INTERNALS\_OF\_STRUCTS is defined. This allows you to freely access the contents of an extended gadget structure from within your extended gadget callback functions.

```
typedef struct {
    UInt16 id;
    FormGadgetAttrType attr;
    RectangleType rect;
    const void *data;
    FormGadgetHandlerType *handler;
} FormGadgetTypeInCallback;
```

## Field Descriptions

id	ID of the gadget resource.
attr	See <a href="#">FormGadgetAttrType</a> .
rect	Location and size of the object.
data	Pointer to any specific data that needs to be stored.
handler	Pointer to a callback function that controls the gadget's behavior and responds to events.

Many form functions ([FrmGetObjectIndex](#), [FrmHideObject](#), and [FrmGetObjectBounds](#), for example) take an object index as one of their arguments. The most common way to get an object's index is to call [FrmGetObjectIndex](#). FrmGetObjectIndex takes a form ID and returns the form object's index. This is the routine one should use in most cases, because the application usually knows the object ID. However, extended gadgets have APIs with callbacks that pass back the gadget pointer and not the ID. In those cases, the only way to get the object index (so one can use the FrmGetObject\* APIs) is to use the function [FrmGetObjectIndexFromPtr](#).

If you need the same functionality on pre-Palm OS 4.0 systems then you can accomplish the same thing with the following code snippet.

---

```
UInt16 index;
UInt16 objIndex = frmInvalidObjectId;
UInt16 numObjects = FrmGetNumberOfObjects(frmP)
for (index = 0; index < numObjects; index++) {
    if (FrmGetObjectPtr(index) == myObjPtr) {
        // Found it
        objIndex = index;
        break;
    }
}
```

---

**Compatibility** Introduced in the Palm OS 4.0 SDK Update 1.

## FormLabelType

The FormLabelType structure defines a label that appears on a form.

## Forms

### Form Data Structures

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `FormLabelType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    UInt16           id;
    PointType        pos;
    FormObjAttrType attr;
    FontID           fontID;
    UInt8            reserved;
    Char             *text;
} FormLabelType;
```

Your code should treat the `FormLabelType` structure as opaque. Do not attempt to change structure member values directly.

#### Field Descriptions

<code>id</code>	Resource ID of the label.
<code>pos</code>	Location of the label.
<code>attr</code>	See <a href="#">FormObjAttrType</a> .
<code>fontID</code>	Font ID of the font used for the label.
<code>reserved</code>	Reserved for future use.
<code>text</code>	Text of the label.

## FormLineType

The `FormLineType` structure defines a line appearing on a form.

```
typedef struct {
    FormObjAttrType   attr;
    PointType         point1;
    PointType         point2;
} FormLineType;
```

Your code should treat the `FormLineType` structure as opaque. Do not attempt to change structure member values directly.

### Field Descriptions

attr	See <a href="#">FormObjAttrType</a> .
point1	Starting point of the line.
point2	Ending point of the line.

## FormObjAttrType

The `FormObjAttrType` bit field defines a form object's attributes.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `FormObjAttrType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    UInt16 usable : 1;
    UInt16 reserved : 15;
} FormObjAttrType;
```

Your code should treat the `FormObjAttrType` structure as opaque. Do not attempt to change structure member values directly.

### Field Descriptions

usable	Not set if the object is not considered part of the current interface of the application, and it doesn't appear on screen.
reserved	Reserved for future use.

## FormObjectKind

The `FormObjectKind` enum specifies values for the `objectType` field of the [FormObjListType](#). It specifies how to interpret the object field.

## Forms

### *Form Data Structures*

---

```
enum formObjects {
    frmFieldObj,
    frmControlObj,
    frmListObj,
    frmTableObj,
    frmBitmapObj,
    frmLineObj,
    frmFrameObj,
    frmRectangleObj,
    frmLabelObj,
    frmTitleObj,
    frmPopupObj,
    frmGraffitiStateObj,
    frmGadgetObj,
    frmScrollbarObj,
};

typedef enum formObjects FormObjectKind;
```

### **Value Descriptions**

frmFieldObj	Text field
frmControlObj	Control
frmListObj	List
frmTableObj	Table
frmBitmapObj	Form bitmap
frmLineObj	Line
frmFrameObj	Frame
frmRectangleObj	Rectangle
frmLabelObj	Label
frmTitleObj	Form title
frmPopupObj	Popup list
frmGraffitiStateObj	Graffiti® state indicator

frmGadgetObj	Gadget (custom object)
frmScrollbarObj	Scrollbar

## FormObjectType

The FormObjectType union points to the C structure for a user interface object that appears on the form.

```
typedef union {
    void                      *ptr;
    FieldType                *field;
    ControlType               *control;
    GraphicControlType        *graphicControl;
    SliderControlType         *sliderControl;
    ListType                  *list;
    TableType                 *table;
    FormBitmapType            *bitmap;
    FormLabelType              *label;
    FormTitleType              *title;
    FormPopupType              *popup;
    FormGraffitiStateType     *grfState;
    FormGadgetType             *gadget;
    ScrollBarType              *scrollBar;
} FormObjectType;
```

Your code should treat the FormObjectType structure as opaque. Do not attempt to change structure member values directly.

### Field Descriptions

ptr	Used when the object's type is not one of those specified below.
field	Text field's structure. See <a href="#">FieldType</a> .
control	Control's structure. See <a href="#">ControlType</a> .
graphicControl	Graphic button structure. See <a href="#">GraphicControlType</a> .
sliderControl	Slider control structure. See <a href="#">SliderControlType</a> .

## Forms

### Form Data Structures

---

list	List object's structure. See <a href="#">ListType</a> .
table	Table structure. See <a href="#">TableType</a> .
bitmap	Form bitmap's structure. See <a href="#">FormBitmapType</a> .
label	Label's structure. See <a href="#">FormLabelTextType</a> .
title	Form title's structure. See <a href="#">FormTitleType</a> .
popup	Popup list's structure. See <a href="#">FormPopupType</a> .
grfState	Graffiti shift indicator's structure. See <a href="#">FrmGraffitiStateType</a> .
gadget	Gadget (custom UI resource) structure. See <a href="#">FormGadgetType</a> .
scrollbar	Scroll bar's structure. See <a href="#">ScrollBarType</a> .

### Compatibility

The `graphicControl` and `sliderControl` fields are only defined if [3.5 New Feature Set](#) is present.

## FormObjListType

The `FormObjectListType` structure specifies a user interface object that appears on the form.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `FormObjListType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    FormObjectKind    objectType;
    UInt8             reserved;
    FormObjectType    object;
} FormObjListType;
```

Your code should treat the `FormObjListType` structure as opaque. Do not attempt to change structure member values directly.

### Field Descriptions

objectType      Specifies the type of the object (control, field, etc.).  
See [FormObjectKind](#).

reserved      Reserved for future use.

object      The C data structure that defines the object. See  
[FormObjectType](#).

## FormPopupType

The FormPopupType structure defines a popup list that appears on a form.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the FormPopupType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

## Forms

### *Form Data Structures*

---

```
typedef struct {
    UInt16           controlID;
    UInt16           listID;
} FormPopupType;
```

Your code should treat the FormPopupType structure as opaque. Do not attempt to change structure member values directly.

#### Field Descriptions

- |           |  |
|-----------|--|
| controlID | Resource ID of the popup trigger control that triggers the list's display. |
| listID    | Resource ID of the list object that defines the popup list.                |

## FormPtr

The FormPtr type defines a pointer to a [FormType](#) structure.

```
typedef FormType *FormPtr;
```

## FormRectangleType

The FormRectangleType structure defines a rectangle that appears on the form.

```
typedef struct {
    FormObjAttrType attr;
    RectangleType   rect;
} FormRectangleType;
```

Your code should treat the FormRectangleType structure as opaque. Do not attempt to change structure member values directly.

#### Field Descriptions

- |      |                                       |
|------|---------------------------------------|
| attr | See <a href="#">FormObjAttrType</a> . |
| rect | Location and size of the rectangle.   |

## FormTitleType

The FormTitleType structure defines the title of the form.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the FormTitleType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    RectangleType rect;
    char           *text;
} FormTitleType;
```

Your code should treat the FormTitleType structure as opaque. Do not attempt to change structure member values directly.

### Field Descriptions

rect	The location and size of the title area.
text	Text of the title.

## FormType

The FormType structure and supporting structures are defined below.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the FormType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    WindowType          window;
    UInt16              formId;
    FormAttrType        attr;
    WinHandle           bitsBehindForm;
    FormEventHandlerType *handler;
}
```

## Forms

### Form Data Structures

---

```
    UInt16          focus;
    UInt16          defaultButton;
    UInt16          helpRscId;
    UInt16          menuRscId;
    UInt16          numObjects;
    *objects;
} FormType;
```

Your code should treat the `FormType` structure as opaque. Do not attempt to change structure member values directly.

#### Field Descriptions

window	Structure of the window object that corresponds to the form. See <a href="#">WindowType</a> . Access this field with <a href="#">FrmGetWindowHandle</a> .
formId	ID number of the form, specified by the application developer. This ID value is part of the event data of <a href="#">frmOpenEvent</a> . The ID should match the form's resource ID. Access this field with <a href="#">FrmGetFormId</a> .
attr	Form object attributes. See <a href="#">FormAttrType</a> .
bitsBehindForm	Used to save all the bits behind the form so the screen can be properly refreshed when the form is closed. This field is for internal use only by modal forms.
handler	Routine called when the form needs to handle an event. You typically set this in your application's event handling function by calling <a href="#">FrmSetEventHandler</a> .
focus	Index of a field or table object within the form that contains the focus. Any <a href="#">keyDownEvent</a> is passed to the object that has the focus. Set to noFocus if no object has the focus. Set this field with <a href="#">FrmSetFocus</a> .

defaultButton	Resource ID of the object defined as the default button. This value is used by the routine <a href="#">FrmDoDialog</a> .
helpRscId	Resource ID number of the help resource. The help resource is a String resource (type tSTR).
menuRscId	ID number of a menu bar to use if the form has a menu, or zero if the form doesn't have a menu.
numObjects	Number of objects contained within the form. Access this field with <a href="#">FrmGetNumberOfObjects</a> .
objects	Pointer to the array of objects contained within the form. See <a href="#">FormObjListType</a> .

## FrmGraffitiStateType

The FrmGraffitiStateType structure defines the graffiti shift indicator.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the FrmGraffitiStateType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct{
    PointerType    pos;
}FrmGraffitiStateType;
```

Your code should treat the FrmGraffitiStateType structure as opaque. Do not attempt to change structure member values directly.

### Field Descriptions

pos              Location of the graffiti shift indicator.

## Forms

### *Form Constants*

---

## Form Constants

The following form constants are defined:

Constant	Value	Description
noFocus	0xffff	No form object has the focus
frmRedrawUpdateCode	0x8000	Indicates that the form should be redrawn; flag in a <a href="#">frmUpdateEvent</a> .
frmNoSelectedControl	0xff	Returned by <a href="#">FrmGetControlGroupSelection</a> if no control is selected.
frmResponseCreate	1974	Passed to <a href="#">FormCheckResponseFuncType</a> to indicate that the function should perform initialization.
frmResponseQuit	0xBEEF	Passed to <a href="#">FormCheckResponseFuncType</a> to indicate that the function should perform cleanup.

---

## Form Resources

The following resources are associated with forms and with the objects on a form whose data structures are defined above:

- Form—[Form Resource](#) (tFRM)
- Alert dialog—[Alert Resource](#) (Talt)
- Bitmap—[Form Bitmap Resource](#) (tFBM)
- Button—[Button Resource](#) (tBTN)
- Check box—[Check Box Resource](#) (tCBX)
- Field—[Field Resource](#) (tFLD)
- Gadget (custom object)—[Gadget Resource](#) (tGDT)
- Graffiti shift indicator—[Graffiti Shift Indicator Resource](#) (tGSI)
- Label—[Label Resource](#) (tLBL)

- List—[List Resource](#) (tLST)
- Popup trigger—[Popup Trigger Resource](#) (tPUT)
- Push button—[Push Button Resource](#) (tPBN)
- Repeating button—[Repeating Button Resource](#) (tREP)
- Scrollbar—[Scroll Bar Resource](#) (tSCL)
- Selector trigger—[Selector Trigger Resource](#) (tSLT)
- Table—[Table Resource](#) (tTBL)

## Form Functions

### FrmAlert

**Purpose** Create a modal dialog from an alert resource and display it until the user selects a button in the dialog.

**Declared In** Form.h

**Prototype** UInt16 FrmAlert (UInt16 alertId)

**Parameters** -> alertId ID of the alert resource.

**Result** Returns the item number of the button the user selected. A button's item number is determined by its order in the alert dialog; the first button has the item number 0 (zero).

---

**NOTE:** A default button press is simulated if the user switches to a different application while a modal dialog is active.

---

**See Also** [FrmDoDialog](#), [FrmCustomAlert](#), [FrmCustomResponseAlert](#)

## Forms

### *Form Functions*

---

## FrmCloseAllForms

**Purpose** Send a [frmCloseEvent](#) to all open forms.

**Declared In** Form.h

**Prototype** void FrmCloseAllForms (void)

**Parameters** None.

**Result** Returns nothing.

**Comments** Applications can call this function to ensure that all forms are closed cleanly before exiting [PilotMain](#); that is, before termination.

**See Also** [FrmSaveAllForms](#)

## FrmCopyLabel

**Purpose** Copy the passed string into the data structure of the specified label object in the active form.

**Declared In** Form.h

**Prototype** void FrmCopyLabel (FormType \*formP,  
                  UInt16 labelID, const Char \*newLabel)

**Parameters**

-> formP	Pointer to the form object ( <a href="#">FormType</a> structure).
-> labelID	ID of form label object.
-> newLabel	Pointer to a null-terminated string.

**Result** Returns nothing.

**Comments** The size of the new label **must not** exceed the size of the label defined in the resource. When defining the label in the resource, specify an initial size at least as big as any of the strings that will be

assigned dynamically. This function redraws the label if the form's `usable` attribute and the label's `visible` attribute are set.

This function redraws the label but does not erase the old one first. If the new label is shorter than the old one, the end of the old label will still be visible. To avoid this, you can hide the label using [FrmHideObject](#), then show it using [FrmShowObject](#), after using `FrmCopyLabel`.

Note that `FrmCopyLabel` copies the passed string into memory already allocated for the label. Thus, the string doesn't need to remain in existence once `FrmCopyLabel` returns.

**See Also** [FrmGetLabel](#)

## **FrmCopyTitle**

**Purpose** Copy a new title over the form's current title. If the form is visible, the new title is drawn.

**Declared In** `Form.h`

**Prototype** `void FrmCopyTitle (FormType *formP,  
const Char *newTitle)`

**Parameters** `-> formP` Pointer to the form object ([FormType](#) structure).  
`-> newTitle` Pointer to the new title string.

**Result** Returns nothing.

**Comments** The size of the new title **must not** exceed the title size defined in the resource. When defining the title in the resource, specify an initial size at least as big as any of the strings to be assigned dynamically.

**See Also** [FrmGetTitle](#), [FrmSetTitle](#)

## Forms

### *Form Functions*

---

## FrmCustomAlert

**Purpose** Create a modal dialog from an alert resource and display the dialog until the user taps a button in the alert dialog.

**Declared In** Form.h

**Prototype** `UInt16 FrmCustomAlert (UInt16 alertId,  
const Char *s1, const Char *s2, const Char *s3)`

**Parameters** `-> alertId` Resource ID of the alert.

`-> s1, s2, s3` Strings to replace ^1, ^2, and ^3 (see Comments).

**Result** Returns the number of the button the user tapped (the first button is zero).

**Comments** A button's item number is determined by its order in the alert template; the first button has the item number zero.

Up to three strings can be passed to this routine. They are used to replace the variables ^1, ^2 and ^3 that are contained in the message string of the alert resource.

If the variables ^1, ^2, and ^3 occur in the message string, do not pass NULL for the arguments s1, s2, and s3. If you want an argument to be ignored, pass the empty string (""). In Palm OS 2.0 or below, pass a string containing a space (" ") instead of the empty string.

---

**NOTE:** A default button press is simulated if the user switches to a different application while a modal dialog is active.

---

**See Also** [FrmAlert](#), [FrmDoDialog](#), [FrmCustomResponseAlert](#)

## FrmCustomResponseAlert

**Purpose** Create a modal dialog with a text field from an alert resource and display it until the user taps a button in the alert dialog.

**Declared In** Form.h

**Prototype** `UInt16 FrmCustomResponseAlert (UInt16 alertId,  
const Char *s1, const Char *s2, const Char *s3,  
Char *entryStringBuf, Int16 entryStringBufLength,  
FormCheckResponseFuncPtr callback)`

**Parameters**

-> alertId	Resource ID of the alert.
-> s1, s2, s3	Strings to replace ^1, ^2, and ^3. See the Comments in <a href="#">FrmCustomAlert</a> for more information.
<- entryStringBuf	The string the user entered in the text field.
-> entryStringBufLength	The maximum length for the string in entryStringBuf.
-> callback	A callback function that processes the string. See <a href="#">FormCheckResponseFuncType</a> . Pass NULL if there is no callback.

**Result** Returns the number of the button the user tapped (the first button is zero).

**Comments** This function differs from [FrmCustomAlert](#) in these ways:

- The dialog it displays contains a text field for user entry. The text that the user enters is returned in the entryStringBuf parameter.
- When the user taps a button, the callback function is called and is passed the button number and entryStringBuf. The dialog is only dismissed if the callback returns true. This behavior allows you to perform error checking on the

## Forms

### *Form Functions*

---

string that the user entered and give the user a chance to re-enter the string.

The callback function is also called with special constants when the alert dialog is being initialized and when it is being deallocated. This allows the callback to perform any necessary initialization and cleanup.

---

**NOTE:** A default button press is simulated if the user switches to a different application while a modal dialog is active.

---

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [FrmAlert](#), [FrmDoDialog](#)

## FrmDeleteForm

**Purpose** Release the memory occupied by a form. Any memory allocated to objects in the form is also released.

**Declared In** Form.h

**Prototype** void FrmDeleteForm (FormType \*formP)

**Parameters** -> formP Pointer to the form object ([FormType](#) structure).

**Result** Returns nothing.

**Comments** This function doesn't modify the display.

**Compatibility** If [3.5 New Feature Set](#) is present and the form contains an extended gadget, this function calls the gadget's callback with formGadgetDeleteCmd. See [FormGadgetHandlerType](#).

**See Also** [FrmInitForm](#), [FrmReturnToForm](#)

## FrmDispatchEvent

<b>Purpose</b>	Dispatch an event to the application's handler for the form.
<b>Declared In</b>	Form.h
<b>Prototype</b>	Boolean FrmDispatchEvent (EventType *eventP)
<b>Parameters</b>	-> eventP      Pointer to an event.
<b>Result</b>	Returns the Boolean value returned by the form's event handler or <a href="#">FrmHandleEvent</a> . (If the form's event handler returns <code>false</code> , the event is passed to <code>FrmHandleEvent</code> .) This function also returns <code>false</code> if the form specified in the event is invalid.
<b>Comments</b>	The event is dispatched to the current form's handler unless the form ID is specified in the event data, as, for example, with <a href="#">frmOpenEvent</a> or <a href="#">frmGotoEvent</a> . A form's event handler ( <a href="#">FormEventHandlerType</a> ) is registered by <a href="#">FrmSetEventHandler</a> .  Note that if the form does not have a registered event handler, this function causes a fatal error.

## FrmDoDialog

<b>Purpose</b>	Display a modal dialog until the user taps a button in the dialog.
<b>Declared In</b>	Form.h
<b>Prototype</b>	UInt16 FrmDoDialog (FormType *formP)
<b>Parameters</b>	-> formP      Pointer to the form object ( <a href="#">FormType</a> structure).

**Result** Returns the resource ID of the button the user tapped.

## Forms

### *Form Functions*

---

**NOTE:** A default button press is simulated if the user switches to a different application while a modal dialog is active.

---

**Comments** Before calling FrmDoDialog you must have called [FrmInitForm](#) to load and initialize the dialog and you must have then set the event handler, if one is needed. After the call, read any values needed from the dialog's objects and then call [FrmDeleteForm](#) to release the memory occupied by the dialog.

**See Also** [FrmInitForm](#), [FrmCustomAlert](#), [FrmCustomResponseAlert](#)

## FrmDrawForm

**Purpose** Draw all objects in a form and the frame around the form.

**Declared In** Form.h

**Prototype** void FrmDrawForm (FormType \*formP)

**Parameters** -> formP Pointer to the form object ([FormType](#) structure).

**Result** Returns nothing.

**Comments** If the saveBehind form attribute is set and the form is visible, this function saves the bits behind the form using the bitsBehindForm field in the FormType structure.

You should call this function in response to a [frmOpenEvent](#).

If you do any custom drawing, you should do so after you call this function not before. If you do custom drawing, respond to [frmUpdateEvent](#) as well as [frmOpenEvent](#), and be sure to return true to specify that the [frmUpdateEvent](#) was handled. The default event handler for [frmUpdateEvent](#) calls [FrmDrawForm](#), so if you allow the event to fall through by returning false, your custom drawing is erased.

<b>Compatibility</b>	If <a href="#">3.5 New Feature Set</a> is present, <code>FrmDrawForm</code> erases the form's window before performing any drawing. Thus, it is especially important to do any custom drawing after this function call on Palm OS 3.5 and higher.
	If <a href="#">3.5 New Feature Set</a> is present and the form contains an extended gadget, this function calls the gadget's callback with <code>formGadgetDrawCmd</code> . See <a href="#">FormGadgetHandlerType</a> .

**See Also** [FrmEraseForm](#), [FrmInitForm](#)

## **FrmEraseForm**

<b>Purpose</b>	Erase a form from the display.
<b>Declared In</b>	<code>Form.h</code>
<b>Prototype</b>	<code>void FrmEraseForm (FormType *formP)</code>
<b>Parameters</b>	<code>-&gt; formP</code> Pointer to the form object ( <a href="#">FormType</a> structure).
<b>Result</b>	Returns nothing.
<b>Comments</b>	If the region obscured by the form was saved by <a href="#">FrmDrawForm</a> , this function restores that region.

## Forms

### *Form Functions*

---

## FrmGetActiveField

<b>Purpose</b>	Return the active field for a specified form.
<b>Declared In</b>	Form.h
<b>Prototype</b>	<code>FieldType *FrmGetActiveField (const FormType *formP)</code>
<b>Parameters</b>	<code>-&gt; formP</code> Pointer to the form for which the active field should be returned, or NULL if the active field on the active form is desired.
<b>Result</b>	Returns a pointer to the field object of the active field, or NULL if the form doesn't have an active field or if there is no active form.
<b>Comments</b>	This function will most often be called with a NULL parameter to obtain the active field on the active form.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">FrmGetActiveForm</a>

## FrmGetActiveForm

<b>Purpose</b>	Return the currently active form.
<b>Declared In</b>	Form.h
<b>Prototype</b>	<code>FormType *FrmGetActiveForm (void)</code>
<b>Parameters</b>	None.
<b>Result</b>	Returns a pointer to the form object of the active form.
<b>Comments</b>	You should not call the <code>FrmGetActiveForm</code> function when a popup window is open. There is no active form while a popup is

displayed, and the value returned from `FrmGetActiveForm` in this situation has no meaning.

**See Also** [FrmGetActiveField](#), [FrmGetActiveFormID](#),  
[FrmSetActiveForm](#)

## **FrmGetActiveFormID**

**Purpose** Return the ID of the currently active form.

**Declared In** Form.h

**Prototype** `UInt16 FrmGetActiveFormID (void)`

**Parameters** None.

**Result** Returns the active form's ID number.

**See Also** [FrmGetActiveForm](#)

## **FrmGetControlGroupSelection**

**Purpose** Return the item number of the control selected in a group of controls.

**Declared In** Form.h

**Prototype** `UInt16 FrmGetControlGroupSelection  
(const FormType *formP, UInt8 groupNum)`

**Parameters** `-> formP` Pointer to the form object ([FormType](#) structure).  
`-> groupNum` Control group number.

**Result** Returns the item number of the selected control; returns `frmNoSelectedControl` if no item is selected.

## Forms

### Form Functions

---

**Comments** The item number is the index into the form object's data structure.

---

**NOTE:** `FrmSetControlGroupSelection` sets the selection in a control group based on an object ID, **not** its index, which `FrmGetControlGroupSelection` returns.

---

**Compatibility** On versions prior to 3.5, this function returned a `Byte` instead of `UInt16`.

**See Also** [FrmGetObjectID](#), [FrmGetObjectPtr](#),  
[FrmSetControlGroupSelection](#)

## FrmGetControlValue

**Purpose** Return the current value of a control.

**Declared In** `Form.h`

**Prototype** `Int16 FrmGetControlValue (const FormType *formP,  
                  UInt16 objIndex)`

**Parameters** `-> formP` Pointer to the form object (`FormType` structure).  
`-> objIndex` Index of the control object in the form object's data structure. You can obtain this by using [FrmGetObjectIndex](#).

**Result** Returns the current value of the control. For most controls the return value is either 0 (off) or 1 (on). For sliders, this function returns the value of the `value` field.

**Comments** The caller must specify a valid index. This function is valid only for push button and check box control objects.

**See Also** [FrmSetControlValue](#)

## FrmGetFirstForm

**Purpose** Return the first form in the window list.

**Declared In** Form.h

**Prototype** FormType \*FrmGetFirstForm (void)

**Parameters** None.

**Result** Returns a pointer to a form object, or NULL if there are no forms.

**Comments** The window list is a LIFO stack. The last window created is the first window in the window list.

## FrmGetFocus

**Purpose** Return the item (index) number of the object that has the focus.

**Declared In** Form.h

**Prototype** UInt16 FrmGetFocus (const FormType \*formP)

**Parameters** -> formP Pointer to the form object ([FormType](#) structure).

**Result** Returns the index of the object (UI element) that has the focus, or returns noFocus if none does. To convert the object index to an ID, use [FrmGetObjectID](#).

**See Also** [FrmGetObjectPtr](#), [FrmSetFocus](#)

## Forms

### *Form Functions*

---

## FrmGetFormBounds

<b>Purpose</b>	Return the visual bounds of the form; the region returned includes the form's frame.					
<b>Declared In</b>	Form.h					
<b>Prototype</b>	<code>void FrmGetFormBounds (const FormType *formP, RectangleType *rP)</code>					
<b>Parameters</b>	<table><tr><td>-&gt; formP</td><td>Pointer to the form object (<a href="#">FormType</a> structure).</td></tr><tr><td>&lt;- rP</td><td>Pointer to a RectangleType structure where the bounds is returned.</td></tr></table>		-> formP	Pointer to the form object ( <a href="#">FormType</a> structure).	<- rP	Pointer to a RectangleType structure where the bounds is returned.
-> formP	Pointer to the form object ( <a href="#">FormType</a> structure).					
<- rP	Pointer to a RectangleType structure where the bounds is returned.					
<b>Result</b>	Returns nothing. The bounds of the form are returned in r.					

## FrmGetFormId

<b>Purpose</b>	Return the resource ID of a form.			
<b>Declared In</b>	Form.h			
<b>Prototype</b>	<code>UInt16 FrmGetFormId (const FormType *formP)</code>			
<b>Parameters</b>	<table><tr><td>-&gt; formP</td><td>Pointer to the form object (<a href="#">FormType</a> structure).</td></tr></table>		-> formP	Pointer to the form object ( <a href="#">FormType</a> structure).
-> formP	Pointer to the form object ( <a href="#">FormType</a> structure).			
<b>Result</b>	Returns form resource ID.			
<b>See Also</b>	<a href="#">FrmGetFormPtr</a>			

## FrmGetFormPtr

**Purpose** Return a pointer to the form that has the specified ID.

**Declared In** Form.h

**Prototype** FormType \*FrmGetFormPtr (UInt16 formId)

**Parameters** -> formId Form ID number.

**Result** Returns a pointer to the form object, or NULL if the form is not in memory.

**See Also** [FrmGetFormId](#)

## FrmGetGadgetData

**Purpose** Return the value stored in the data field of the gadget object.

**Declared In** Form.h

**Prototype** void \*FrmGetGadgetData (const FormType \*formP,  
                  UInt16 objIndex)

**Parameters** -> formP Pointer to the form object ([FormType](#) structure).

-> objIndex Index of the gadget object in the form object's data structure. You can obtain this by using [FrmGetObjectIndex](#).

**Result** Returns a pointer to the custom gadget's data.

**Comments** Gadget objects provide a way for an application to attach custom gadgetry to a form. In general, the data field of a gadget object contains a pointer to the custom object's data structure.

**See Also** [FrmSetGadgetData](#), [FrmSetGadgetHandler](#)

## Forms

### *Form Functions*

---

## FrmGetLabel

**Purpose** Return pointer to the text of the specified label object in the specified form.

**Declared In** Form.h

**Prototype** const Char \*FrmGetLabel (const FormType \*formP,  
                  UInt16 labelID)

**Parameters** -> formP                 Pointer to the form object ([FormType](#) structure).

-> labelID                 ID of the label object.

**Result** Returns a pointer to the label string.

**Comments** Does not make a copy of the string; returns a pointer to the string. The object must be a label.

**See Also** [FrmCopyLabel](#)

## FrmGetNumberOfObjects

**Purpose** Return the number of objects in a form.

**Declared In** Form.h

**Prototype** UInt16 FrmGetNumberOfObjects  
                  (const FormType \*formP)

**Parameters** -> formP                 Pointer to the form object ([FormType](#) structure).

**Result** Returns the number of objects in the specified form.

**See Also** [FrmGetObjectPtr](#), [FrmGetObjectID](#)

## FrmGetObjectBounds

**Purpose** Retrieve the bounds of an object given its form and index.

**Declared In** Form.h

**Prototype** void FrmGetObjectBounds (const FormType \*formP,  
                  UInt16 objIndex, RectangleType \*rP)

**Parameters**

-> formP	Pointer to the form object ( <a href="#">FormType</a> structure).
-> objIndex	Index of an object in the form. You can obtain this by using <a href="#">FrmGetObjectIndex</a> .
<- rP	Pointer to a RectangleType structure where the object bounds are returned. The bounds are in window-relative coordinates.

**Result** Returns nothing. The object's bounds are returned in r.

**See Also** [FrmGetObjectPosition](#), [FrmSetObjectPosition](#)

## FrmGetObjectId

**Purpose** Return the ID of the specified object.

**Declared In** Form.h

**Prototype** UInt16 FrmGetObjectId (const FormType \*formP,  
                  UInt16 objIndex)

**Parameters**

-> formP	Pointer to the form object ( <a href="#">FormType</a> structure).
----------	---

## Forms

### *Form Functions*

---

-> objIndex      Index of an object in the form. You can obtain this by using [FrmGetObjectIndex](#).

**Result**      Returns the ID number of an object or `frmInvalidObjectID` if the `objIndex` parameter is invalid.

**See Also**      [FrmGetObjectPtr](#)

## FrmGetObjectIndex

**Purpose**      Return the index of an object in the form's objects list.

**Declared In**      `Form.h`

**Prototype**      `UInt16 FrmGetObjectIndex (const FormType *formP,  
                  UInt16 objID)`

**Parameters**      -> `formP`      Pointer to the form object ([FormType](#) structure).  
                  -> `objID`      ID of an object in the form.

**Result**      Returns the index of the specified object (the index of the first object is 0), or `frmInvalidObjectID` if the supplied object ID is invalid.

**Comments**      Bitmaps use a different mechanism for IDs than the rest of the form objects. When finding a bitmap with `FrmGetObjectIndex`, you need to pass the bitmap's resource ID, not the ID of the form bitmap object. (Passing the ID of the form bitmap object may or may not give you the right object back, depending on how you created the objects.)

This means that if you've got the same bitmap in two different form bitmap objects on the same form, you won't be able to use `FrmGetObjectIndex` to get at the second one; it'll always return the first.

**See Also**      [FrmGetObjectPtr](#), [FrmGetObjectID](#)

## FrmGetObjectIndexFromPtr

**Purpose** Return an object's index.

**Declared In** Form.h

**Prototype** `UInt16 FrmGetObjectIndexFromPtr  
(const FormType *formP, void *objP)`

**Parameters** `-> formP` Pointer to a [FormType](#).  
`-> objP` Pointer to an object.

**Result** Returns the object's index. `frmInvalidObjectId` is returned if `objP` is not associated with the form.

**Comments** Many form functions ([FrmGetObjectType](#), [FrmHideObject](#), and [FrmGetObjectBounds](#), for example) take an object index as one of their arguments. The most common way to get an object's index is to call [FrmGetObjectIndex](#). `FrmGetObjectIndex` takes a form ID and returns the form object's index. This is the routine one should use in most cases, because the application usually knows the object ID. However, gadgets and specifically extended gadgets, have APIs with callbacks that pass back the gadget pointer and not the ID. In those cases, the only way to get the object index (so one can use the `FrmGetObject*` APIs) is to use `FrmGetObjectIndexFromPtr`.

If you need the same functionality on pre-Palm OS 4.0 systems then you can accomplish the same thing with the following code snippet.

---

```
UInt16 index;  
UInt16 objIndex = frmInvalidObjectId;  
UInt16 numObjects = FrmGetNumberOfObjects(formP)  
for (index = 0; index < numObjects; index++) {  
    if (FrmGetObjectPtr(index) == myObjPtr) {  
        // Found it  
        objIndex = index;  
        break;  
    }  
}
```

---

## Forms

### *Form Functions*

---

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## FrmGetObjectPosition

**Purpose** Return the coordinates of the specified object relative to the form.

**Declared In** Form.h

**Prototype** void FrmGetObjectPosition (const FormType \*formP,  
                  UInt16 objIndex, Coord \*x, Coord \*y)

**Parameters**

-> formP	Pointer to the form object ( <a href="#">FormType</a> structure).
-> objIndex	Index of an object in the form. You can obtain this by using <a href="#">FrmGetObjectIndex</a> .
<- x, y	Pointers where the window-relative x and y positions of the object are returned. These locate the top-left corner of the object.

**Result** Returns nothing.

**See Also** [FrmGetObjectBounds](#), [FrmSetObjectPosition](#)

## FrmGetObjectPtr

**Purpose** Return a pointer to the data structure of an object in a form.

**Declared In** Form.h

**Prototype** void \*FrmGetObjectPtr (const FormType \*formP,  
                  UInt16 objIndex)

**Parameters**

-> formP	Pointer to the form object ( <a href="#">FormType</a> structure).
----------	---

-> objIndex      Index of an object in the form. You can obtain this by using [FrmGetObjectIndex](#).

**Result**      Returns a pointer to an object in the form.

**See Also**      [FrmGetObjectID](#)

## FrmGetObjectType

**Purpose**      Return the type of an object.

**Declared In**      Form.h

**Prototype**      FormObjectKind FrmGetType  
(const FormType \*formP, UInt16 objIndex)

**Parameters**      -> formP      Pointer to the form object ([FormType](#) structure).

-> objIndex      Index of an object in the form. You can obtain this by using [FrmGetObjectIndex](#).

**Result**      Returns FormObjectKind of the item specified. See [FormObjectKind](#).

## FrmGetTitle

**Purpose**      Return a pointer to the title string of a form.

**Declared In**      Form.h

**Prototype**      const Char \*FrmGetTitle (const FormType \*formP)

**Parameters**      -> formP      Pointer to the form object ([FormType](#) structure).

**Result**      Returns a pointer to title string, or NULL if there is no title string or there is an error finding it.

## Forms

### Form Functions

---

**Comments** This is a pointer to the internal structure itself, **not** to a copy.

**See Also** [FrmCopyTitle](#), [FrmSetTitle](#)

## FrmGetWindowHandle

**Purpose** Return the window handle of a form.

**Declared In** Form.h

**Prototype** WinHandle FrmGetWindowHandle  
(const FormType \*formP)

**Parameters** -> formP Pointer to the form object ([FormType](#) structure).

**Result** Returns the handle of the memory block that contains the form data structure. Since the form structure begins with the [WindowType](#), this is also a WinHandle.

## FrmGotoForm

**Purpose** Send a [frmCloseEvent](#) to the current form; send a [frmLoadEvent](#) and a [frmOpenEvent](#) to the specified form.

**Declared In** Form.h

**Prototype** void FrmGotoForm (UInt16 formId)

**Parameters** -> formId ID of the form to display.

**Result** Returns nothing.

**Comments** The default form event handler ([FrmHandleEvent](#)) erases and disposes of a form when it receives a [frmCloseEvent](#).

**See Also** [FrmPopupForm](#)

## FrmHandleEvent

**Purpose** Handle the event that has occurred in the form.

**Declared In** Form.h

**Prototype** Boolean FrmHandleEvent (FormType \*formP,  
EventType \*eventP)

**Parameters** -> formP Pointer to the form object ([FormType](#) structure).  
-> eventP Pointer to the event data structure ([EventType](#)).

**Result** Returns true if the event was handled.

**Comments** Never call this function directly. Call [FrmDispatchEvent](#) instead. FrmDispatchEvent passes events to a form's custom event handler and then, if the event was not handled, to this function.

---

**WARNING!** You should never call this function directly. You should call the FrmDispatchEvent function instead.

---

[Table 11.1](#) provides an overview of how FrmHandleEvent handles different events.

## Forms

### *Form Functions*

---

**Table 11.1 FrmHandleEvent Actions**

<b>When FrmHandleEvent receives...</b>	<b>FrmHandleEvent performs these actions...</b>
<a href="#">ctlEnterEvent</a>	Passes the event and a pointer to the object the event occurred in to <a href="#">CtlHandleEvent</a> . The object pointer is obtained from the event data. If the control is part of an exclusive control group, it deselects the currently selected control of the group first.
<a href="#">ctlRepeatEvent</a>	Passes the event and a pointer to the object the event occurred in to CtlHandleEvent. The object pointer is obtained from the event data.
<a href="#">ctlSelectEvent</a>	Checks if the control is a Popup Trigger Control. If it is, the list associated with the popup trigger is displayed until the user makes a selection or touches the pen outside the bounds of the list. If a selection is made, a <a href="#">popSelectEvent</a> is added to the event queue.
<a href="#">fldEnterEvent</a> or <a href="#">fldHeightChangedEvent</a>	Checks if a field object or a table object has the focus and passes the event to the appropriate handler ( <a href="#">FldHandleEvent</a> or <a href="#">TblHandleEvent</a> ). The table object is also a container object, which may contain a field object. If TblHandleEvent receives a field event, it passes the event to the field object contained within it.
<a href="#">frmCloseEvent</a>	Erases the form and releases any memory allocated for it.
<a href="#">frmGadgetEnterEvent</a>	Passes the event to the gadget's callback function if the gadget has one. See <a href="#">FormGadgetHandlerType</a> .
<a href="#">frmGadgetMiscEvent</a>	Passes the event to the gadget's callback function if the gadget has one. See <a href="#">FormGadgetHandlerType</a> .
<a href="#">frmTitleEnterEvent</a>	Tracks the pen until it is lifted. If it is lifted within the bounds of the form title, adds a <a href="#">frmTitleSelectEvent</a> event to the event queue.

**Table 11.1 FrmHandleEvent Actions (*continued*)**

<b>When FrmHandleEvent receives...</b>	<b>FrmHandleEvent performs these actions...</b>
<a href="#"><u>frmTitleSelectEvent</u></a>	Adds a <a href="#"><u>keyDownEvent</u></a> with the vchrMenu character to the event queue.
<a href="#"><u>frmUpdateEvent</u></a>	Calls <a href="#"><u>FrmDrawForm</u></a> to redraw the form.
<a href="#"><u>keyDownEvent</u></a>	Passes the event to the handler for the object that has the focus. If no object has the focus, the event is ignored.
<a href="#"><u>lstEnterEvent</u></a>	Passes the event and a pointer to the object the event occurred in to <a href="#"><u>LstHandleEvent</u></a> . The object pointer is obtained from the event data.
<a href="#"><u>menuCmdBarOpenEvent</u></a>	Checks if a field object or a table object has the focus and passes the event to the appropriate handler (FldHandleEvent or TblHandleEvent), broadcasts the notification <a href="#"><u>sysNotifyMenuBarOpenEvent</u></a> , and then displays the command toolbar.
<a href="#"><u>menuEvent</u></a>	Checks if the menu command is one of the system edit menu commands. The system provides a standard edit menu that contains the commands Undo, Cut, Copy, Paste, Select All, and Keyboard. FrmHandleEvent responds to these commands.
<a href="#"><u>penDownEvent</u></a> ; pen position in the bounds of the form object	Checks the list of objects contained by the form to determine if the pen is within the bounds of one. If it is, the appropriate handler is called to handle the event, for example, if the pen is in a control, CtlHandleEvent is called. If the pen isn't within the bounds of an object, the event is ignored by the form. If the pen is within the bounds of the help icon, it is tracked until it is lifted, and if it's still within the help icon bounds, the help dialog is displayed.
<a href="#"><u>popSelectEvent</u></a>	Sets the label of the popup trigger to the current selection of the popup list.

## Forms

### *Form Functions*

---

**Table 11.1 FrmHandleEvent Actions (*continued*)**

<b>When FrmHandleEvent receives...</b>	<b>FrmHandleEvent performs these actions...</b>
<u>sclEnterEvent</u> or <u>sclRepeatEvent</u>	Passes the event and a pointer to the object the event occurred in to <u>SclHandleEvent</u> .
<u>tblEnterEvent</u>	Passes the event and a pointer to the object the event occurred in to TblHandleEvent. The object pointer is obtained from the event data.

**Compatibility** FrmHandleEvent only handles frmTitleSelectEvent, menuCmdBarOpenEvent, frmGadgetEnterEvent, and frmGadgetMiscEvent if [3.5 New Feature Set](#) is present. If [5.0 New Feature Set](#) is present, this function should be considered “System Use Only”; applications should do what they can to avoid using it.

**See Also** [FrmDispatchEvent](#)

## FrmHelp

**Purpose** Display the specified help message until the user taps the Done button in the help dialog.

**Declared In** Form.h

**Prototype** void FrmHelp (UInt16 helpMsgId)

**Parameters** -> helpMsgId      Resource ID of help message string.

**Result** Returns nothing.

**Comments** The help message is displayed in a modal dialog that supports scrolling the text if necessary.

## FrmHideObject

**Purpose** Erase the specified object and set its attribute data (usable bit) so that it does not redraw or respond to the pen.

**Declared In** Form.h

**Prototype** void FrmHideObject (FormType \*formP,  
                  UInt16 objIndex)

**Parameters** -> formP                   Pointer to the form object ([FormType](#) structure).  
                  -> objIndex               Index of an object in the form. You can obtain this by using [FrmGetObjectIndex](#).

**Result** Returns nothing.

**Compatibility** Prior to OS version 3.2, this function did not set the usable bit of the object attribute data to false. On an OS version prior to 3.2 you can work around this bug by directly setting this bit to false yourself.

On versions of Palm OS prior to 3.5 this function doesn't affect lists or tables. On Palm OS 3.5 it operates correctly on lists but doesn't have any effect on tables. On Palm OS 4.0 it operates correctly on both lists and tables.

If [3.5 New Feature Set](#) is present and the object is an extended gadget, this function calls the gadget's callback with formGadgetEraseCmd. See [FormGadgetHandlerType](#).

**See Also** [FrmShowObject](#)

## Forms

### *Form Functions*

---

## FrmInitForm

**Purpose** Load and initialize a form resource.

**Declared In** Form.h

**Prototype** FormType \*FrmInitForm (UInt16 rscID)

**Parameters** -> rscID              Resource ID of the form.

**Result** Returns a pointer to the form data structure.

When using debug ROMs, FrmInitForm displays an error message if the form has already been initialized.

**Comments** This function does not affect the display (use [FrmDrawForm](#) to draw the form) nor make the form active (use [FrmSetActiveForm](#) to make it active).

For each initialized form, you must call FrmDeleteForm to release the form memory when you are done with the form. Alternatively, you can free the active form by calling FrmReturnToForm.

**See Also** [FrmDoDialog](#), [FrmDeleteForm](#), [FrmReturnToForm](#)

## FrmNewBitmap

**Purpose** Create a new form bitmap dynamically.

**Declared In** Form.h

**Prototype** FormBitmapType \*FrmNewBitmap (FormType \*\*formPP,  
UInt16 ID, UInt16 rscID, Coord x, Coord y)

**Parameters** <-> formPP Pointer to a pointer to the form in which the new bitmap is installed. This value is not a handle; that is, the old formPP value is not necessarily valid after this function returns because the form may be moved in memory. In subsequent calls, always use the new formPP value returned by this function.

-> ID Symbolic ID of the bitmap, specified by the developer. By convention, this ID should match the resource ID (not mandatory).

-> rscID Numeric value identifying the resource that provides the bitmap. This value must be unique within the application scope.

-> x Horizontal coordinate of the upper-left corner of the bitmap's boundaries, relative to the window in which it appears.

-> y Vertical coordinate of the upper-left corner of the bitmap's boundaries, relative to the window in which it appears.

**Result** Returns a pointer to the new bitmap, or 0 if the call did not succeed. The most common cause of failure is lack of memory.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [FrmRemoveObject](#)

## Forms

### *Form Functions*

---

## FrmNewForm

**Purpose** Create a new form object dynamically.

**Declared In** Form.h

**Prototype** FormType \*FrmNewForm (UInt16 formID,  
const Char \*titleStrP, Coord x, Coord y,  
Coord width, Coord height, Boolean modal,  
UInt16 defaultButton, UInt16 helpRscID,  
UInt16 menuRscID)

<b>Parameters</b>	-> formID	Symbolic ID of the form, specified by the developer. By convention, this ID should match the resource ID (not mandatory).
	-> titleStrP	Pointer to a string that is the title of the form.
	-> x	Horizontal coordinate of the upper-left corner of the form's boundaries, relative to the window in which it appears.
	-> y	Vertical coordinate of the upper-left corner of the form's boundaries, relative to the window in which it appears.
	-> width	Width of the form, expressed in pixels. Valid values are 1 -160.
	-> height	Height of the form, expressed in pixels. Valid values are 1 -160.
	-> modal	true specifies that the form ignores pen events outside its boundaries.
	-> defaultButton	Symbolic ID of the button that provides the form's default action, specified by the developer.
	-> helpRscID	Symbolic ID of the resource that provides the form's online help, specified by the developer. Only modal dialogs can have help resources.

-> menuRscID      Symbolic ID of the resource that provides the form's menus, specified by the developer.

**Result**      Returns a pointer to the new form object, or 0 if the call did not succeed. The most common cause of failure is lack of memory.

**Compatibility**      Implemented only if [3.0 New Feature Set](#) is present.

**See Also**      [FrmValidatePtr](#), [WinValidateHandle](#), [FrmRemoveObject](#)

## FrmNewGadget

**Purpose**      Create a new gadget dynamically and install it in the specified form.

**Declared In**      Form.h

**Prototype**      FormGadgetType \*FrmNewGadget (FormType \*\*formPP,  
                  UInt16 id, Coord x, Coord y, Coord width,  
                  Coord height)

**Parameters**      <-> formPP      Pointer to a pointer to the form in which the new gadget is installed. This value is not a handle; that is, the old formPP value is not necessarily valid after this function returns because the form may be moved in memory. In subsequent calls, always use the new formPP value returned by this function.

-> id      Symbolic ID of the gadget, specified by the developer. By convention, this ID should match the resource ID (not mandatory).

-> x      Horizontal coordinate of the upper-left corner of the gadget's boundaries, relative to the window in which it appears.

-> y      Vertical coordinate of the upper-left corner of the gadget's boundaries, relative to the window in which it appears.

## Forms

### *Form Functions*

---

-> width            Width of the gadget, expressed in pixels. Valid values are 1 - 160.

-> height          Height of the gadget, expressed in pixels. Valid values are 1 - 160.

**Result**      Returns a pointer to the new gadget object or 0 if the call did not succeed. The most common cause of failure is lack of memory.

**Comments**     A gadget is a custom user interface object. For more information, see “[Gadget Resource](#)” on page 50.

**Compatibility**   Implemented only if [3.0 New Feature Set](#) is present.

**See Also**       [FrmRemoveObject](#)

## FrmNewGsi

**Purpose**      Create a new Graffiti shift indicator dynamically and install it in the specified form.

**Declared In**    Form.h

**Prototype**     `FrmGraffitiStateType *FrmNewGsi  
(FormType **formPP, Coord x, Coord y)`

**Parameters**    `<-> formPP`      Pointer to a pointer to the form in which the new Graffiti shift indicator is installed. This value is not a handle; that is, the old formPP value is not necessarily valid after this function returns because the form may be moved in memory. In subsequent calls, always use the new formPP value returned by this function.

`-> x`            Horizontal coordinate of the upper-left corner of the Graffiti shift indicator’s boundaries, relative to the window in which it appears.

-> y      Vertical coordinate of the upper-left corner of the Graffiti shift indicator's boundaries, relative to the window in which it appears.

**Result**      Returns a pointer to the new gadget object or 0 if the call did not succeed. The most common cause of failure is lack of memory.

**Comments**      In normal operation, the Graffiti shift indicator is drawn in the lower-right portion of the screen when the user enters the shift keystroke. You use this function if the Graffiti shift indicator needs to be drawn in a nonstandard location. For example, the form manager uses it to draw the shift indicator in a custom alert dialog that contains a text field ([FrmCustomResponseAlert](#)).

**Compatibility**      Implemented only if [3.5 New Feature Set](#) is present.

**See Also**      [FrmRemoveObject](#)

## FrmNewLabel

**Purpose**      Create a new label object dynamically and install it in the specified form.

**Declared In**      Form.h

**Prototype**      `FormLabelType *FrmNewLabel (FormType **formPP,  
                  UInt16 ID, const Char *textP, Coord x, Coord y,  
                  FontID font)`

**Parameters**      `<-> formPP`      Pointer to a pointer to the form in which the new label is installed. This value is not a handle; that is, the old `formPP` value is not necessarily valid after this function returns because the form may be moved in memory. In subsequent calls, always use the new `formPP` value returned by this function.

## Forms

### Form Functions

---

-> ID	Symbolic ID of the label, specified by the developer. By convention, this ID should match the resource ID (not mandatory).
-> textP	Pointer to a string that provides the label text. This string is copied into the label structure.
-> x	Horizontal coordinate of the upper-left corner of the label's boundaries, relative to the window in which it appears.
-> y	Vertical coordinate of the upper-left corner of the label's boundaries, relative to the window in which it appears.
-> font	Font with which to draw the label text.

**Result** Returns a pointer to the new label object or 0 if the call did not succeed. The most common cause of failure is lack of memory.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [CtlValidatePointer](#), [FrmRemoveObject](#)

## FrmPointInTitle

**Purpose** Check if a coordinate is within the bounds of the form's title.

**Declared In** Form.h

**Prototype** Boolean FrmPointInTitle (const FormType \*formP,  
Coord x, Coord y)

**Parameters** -> formP Pointer to the form object ([FormType](#) structure).  
-> x, y Window-relative x and y coordinates.

**Result** Returns true if the specified coordinate is in the form's title.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## FrmPopupForm

**Purpose** Queues a [frmLoadEvent](#) and a [frmOpenEvent](#) for the specified form.

**Declared In** Form.h

**Prototype** void FrmPopupForm (UInt16 formID)

**Parameters** -> formID      Resource ID of form to open.

**Result** Returns nothing.

**Comments** This routine differs from [FrmGotoForm](#) in that the current form is not closed. You can call [FrmReturnToForm](#) to close a form opened by FrmPopupForm.

## FrmRemoveObject

**Purpose** Remove the specified object from the specified form.

**Declared In** Form.h

**Prototype** Err FrmRemoveObject (FormType \*\*formPP,  
                  UInt16 objIndex)

**Parameters** <-> formPP      Pointer to a pointer to the form from which this function removes an object. This value is not a handle; that is, the old formPP value is not necessarily valid after this function returns. In subsequent calls, always use the new formPP value returned by this function.

-> objIndex      The object to remove, specified as an index into the list of objects installed in the form. You can use the [FrmGetObjectIndex](#) function to discover this value.

**Result** Returns 0 if no error.

## Forms

### *Form Functions*

---

<b>Comments</b>	You can use this function to remove any form object (a bitmap, control, list, and so on) and free the memory allocated to it within the form data structure. The data structures for most form objects are embedded within the form data structure memory chunk. This function frees that memory and moves the other objects, if necessary, to close up the memory "hole" and decrease the size of the form chunk.  Note that this function does not free memory outside the form data structure that may be allocated to an object, such as the memory allocated to the string in an editable field object.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">FrmNewBitmap</a> , <a href="#">FrmNewForm</a> , <a href="#">FrmNewGadget</a> , <a href="#">FrmNewLabel</a> , <a href="#">CtlNewControl</a> , <a href="#">FldNewField</a> , <a href="#">LstNewList</a>

## FrmRestoreActiveState

<b>Purpose</b>	Macro that restores the active window and form state.
<b>Declared In</b>	Form.h
<b>Prototype</b>	<code>FrmRestoreActiveState (stateP)</code>
<b>Parameters</b>	<code>-&gt; stateP</code> A pointer to the <code>FormActiveStateType</code> structure that you passed to <code>FrmSaveActiveState</code> when you saved the state.
<b>Result</b>	Returns zero on success.
<b>Comments</b>	Use this function to restore the state of displayed forms to the state that existed before you dynamically showed a new modal form. You must have previously called <a href="#">FrmSaveActiveState</a> to save the state.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## FrmReturnToForm

**Purpose** Erase and delete the currently active form and make the specified form the active form.

**Declared In** Form.h

**Prototype** void FrmReturnToForm (UInt16 formID)

**Parameters** -> formID      Resource ID of the form to return to.

**Result** Returns nothing.

**Comments** It is assumed that the form being returned to is already loaded into memory and initialized. Passing a form ID of 0 returns to the first form in the window list, which is the last form to be loaded.

FrmReturnToForm does not generate a frmCloseEvent when called from a modal form's event handler. It assumes that you have already handled cleaning up your form's variables since you are explicitly calling FrmReturnToForm.

**See Also** [FrmGotoForm](#), [FrmPopupForm](#)

## Forms

### *Form Functions*

---

## FrmSaveActiveState

<b>Purpose</b>	Macro that saves the active window and form state.
<b>Declared In</b>	Form.h
<b>Prototype</b>	<code>FrmSaveActiveState (stateP)</code>
<b>Parameters</b>	<code>&lt;-&gt; stateP</code> A pointer to a <code>FormActiveStateType</code> structure that is used to save the state. Pass the same pointer to <code>FrmRestoreActiveState</code> to restore the state. Treat the structure like a black box; that is, don't attempt to read it or write to it.
<b>Result</b>	Returns zero on success.
<b>Comments</b>	Use this function to save the state of displayed forms before dynamically showing a new modal form. Call <a href="#"><u>FrmRestoreActiveState</u></a> to restore the state after you remove the modal form.
<b>Compatibility</b>	Implemented only if <a href="#"><u>3.0 New Feature Set</u></a> is present.

## FrmSaveAllForms

<b>Purpose</b>	Send a <a href="#"><u>frmSaveEvent</u></a> to all open forms.
<b>Declared In</b>	Form.h
<b>Prototype</b>	<code>void FrmSaveAllForms (void)</code>
<b>Parameters</b>	None.
<b>Result</b>	Returns nothing.
<b>See Also</b>	<a href="#"><u>FrmCloseAllForms</u></a>

## FrmSetActiveForm

<b>Purpose</b>	Set the active form. All input (key and pen) is directed to the active form and all drawing occurs there.
<b>Declared In</b>	Form.h
<b>Prototype</b>	<code>void FrmSetActiveForm (FormType *formP)</code>
<b>Parameters</b>	<code>-&gt; formP</code> Pointer to the form object ( <a href="#">FormType</a> structure).
<b>Result</b>	Returns nothing.
<b>Comments</b>	A <a href="#">penDownEvent</a> outside the form but within the display area is ignored.
<b>Compatibility</b>	In Palm OS releases earlier than 3.5, this function generated a <a href="#">winEnterEvent</a> for the new form immediately following the <a href="#">winExitEvent</a> for the old form. Starting in Palm OS 3.5, <code>FrmSetActiveForm</code> does not generate the <code>winEnterEvent</code> . The <code>winEnterEvent</code> does not occur until the newly active form is drawn.
<b>See Also</b>	<a href="#">FrmGetActiveForm</a>

## FrmSetCategoryLabel

<b>Purpose</b>	Set the category label displayed on the title line of a form. If the form's <code>visible</code> attribute is set, redraw the label.
<b>Declared In</b>	Form.h
<b>Prototype</b>	<code>void FrmSetCategoryLabel (const FormType *formP, UInt16 objIndex, Char *newLabel)</code>
<b>Parameters</b>	<code>-&gt; formP</code> Pointer to the form object ( <a href="#">FormType</a> structure).

## Forms

### *Form Functions*

---

-> objIndex      Index of an object in the form. You can obtain this by using [FrmGetObjectIndex](#).

-> newLabel      Pointer to the name of the new category.

**Result**      Returns nothing.

**Comments**      The pointer to the new label (newLabel) is saved in the object.

## FrmSetControlGroupSelection

**Purpose**      Set the selected control in a group of controls.

**Declared In**      Form.h

**Prototype**      `void FrmSetControlGroupSelection  
(const FormType *formP, UInt8 groupNum,  
UInt16 controlID)`

**Parameters**      -> formP      Pointer to the form object ([FormType](#) structure).

-> groupNum      Control group number.

-> controlID      ID of control to set.

**Result**      Returns nothing.

**Comments**      This function unsets all the other controls in the group. The display is updated.

---

**NOTE:** `FrmGetControlGroupSelection` returns the selection in a control group as an object index, **not** as an object ID, which `FrmSetControlGroupSelection` uses to set the selection.

---

**See Also**      [FrmGetControlGroupSelection](#)

## FrmSetControlValue

**Purpose** Set the current value of a control. If the control is visible, it's redrawn.

**Declared In** Form.h

**Prototype** void FrmSetControlValue (const FormType \*formP,  
                  UInt16 objIndex, Int16 newValue)

**Parameters**

-> formP	Pointer to the form object ( <a href="#">FormType</a> structure).
-> objIndex	Index of the control in the form. You can obtain this by using <a href="#">FrmGetObjectIndex</a> .
-> newValue	New value to set for the control. For sliders, specify a value between the slider's minimum and maximum. For graphical controls, push buttons, or check boxes, specify 0 for off, nonzero for on.

**Result** Returns nothing.

**Comments** This function works only with graphical controls, sliders, push buttons, and check boxes. If you set the value of any other type of control, the behavior is undefined.

**See Also** [FrmGetControlValue](#)

## Forms

### *Form Functions*

---

## FrmSetEventHandler

**Purpose** Registers the event handler callback routine for the specified form.

**Declared In** Form.h

**Prototype** void FrmSetEventHandler (FormType \*formP,  
FormEventHandlerType \*handler)

**Parameters** -> formP Pointer to the form object ([FormType](#) structure).

-> handler Address of the form event handler function, [FormEventHandlerType](#).

**Result** Returns nothing.

**Comments** [FrmDispatchEvent](#) calls this handler whenever it receives an event for a specific form.

`FrmSetEventHandler` must be called right after a form resource is loaded. The callback routine it registers is the mechanism for dispatching events to an application. The tutorial explains how to use callback routines.

## FrmSetFocus

**Purpose** Set the focus of a form to the specified object.

**Declared In** Form.h

**Prototype** void FrmSetFocus (FormType \*formP,  
UInt16 fieldIndex)

**Parameters** -> formP Pointer to the form object ([FormType](#) structure).

-> fieldIndex Index of the object to get the focus in the form.  
You can obtain this by using  
[FrmGetObjectIndex](#). You can pass the  
constant noFocus so that no object has the  
focus.

**Result** Returns nothing.

**Comments** You can set the focus to a field or table object. If the focus is set to a field object, this function turns on the insertion point in the field by calling [FldGrabFocus](#) internally.

**See Also** [FrmGetFocus](#)

## FrmSetGadgetData

**Purpose** Store a data value in the data field of the gadget object.

**Declared In** Form.h

**Prototype** void FrmSetGadgetData (FormType \*formP,  
UInt16 objIndex, const void \*data)

**Parameters**

-> formP	Pointer to the form object ( <a href="#">FormType</a> structure).
-> objIndex	Index of an object in the form. You can obtain this by using <a href="#">FrmGetObjectIndex</a> .
-> data	Application-defined value. This value is stored into the data field of the gadget data structure ( <a href="#">FormGadgetType</a> ).

**Result** Returns nothing.

**Comments** Gadget objects provide a way for an application to attach custom gadgetry to a form. Typically, the data field of a gadget object contains a pointer to the custom object's data structure.

**See Also** [FrmGetGadgetData](#), [FrmSetGadgetHandler](#)

## Forms

### *Form Functions*

---

## FrmSetGadgetHandler

<b>Purpose</b>	Registers the gadget event handler callback routine for the specified gadget on the specified form.						
<b>Declared In</b>	Form.h						
<b>Prototype</b>	<pre>void FrmSetGadgetHandler (FormType *formP,                            UInt16 objIndex, FormGadgetHandlerType *attrP)</pre>						
<b>Parameters</b>	<table><tr><td>-&gt; formP</td><td>Pointer to the form object (<a href="#">FormType</a> structure).</td></tr><tr><td>-&gt; objIndex</td><td>Index of a gadget object in the form. You can obtain this by using <a href="#">FrmGetObjectIndex</a>.</td></tr><tr><td>-&gt; attrP</td><td>Address of the callback function. See <a href="#">FormGadgetHandlerType</a>.</td></tr></table>	-> formP	Pointer to the form object ( <a href="#">FormType</a> structure).	-> objIndex	Index of a gadget object in the form. You can obtain this by using <a href="#">FrmGetObjectIndex</a> .	-> attrP	Address of the callback function. See <a href="#">FormGadgetHandlerType</a> .
-> formP	Pointer to the form object ( <a href="#">FormType</a> structure).						
-> objIndex	Index of a gadget object in the form. You can obtain this by using <a href="#">FrmGetObjectIndex</a> .						
-> attrP	Address of the callback function. See <a href="#">FormGadgetHandlerType</a> .						
<b>Result</b>	Returns nothing.						
<b>Comments</b>	This function sets the application-defined function that controls the specified gadget's behavior. This function is called when the gadget needs to be drawn, erased, deleted, or needs to handle an event.						
<b>Compatibility</b>	Implemented only if <a href="#">3.5 New Feature Set</a> is present.						
<b>See Also</b>	<a href="#">FrmGetGadgetData</a> , <a href="#">FrmSetGadgetData</a>						

## FrmSetMenu

**Purpose** Change a form's menu bar and make the new menu active.

**Declared In** Form.h

**Prototype** void FrmSetMenu (FormType \*formP,  
                  UInt16 menuRscID)

**Parameters** -> formP                   Pointer to the form object ([FormType](#) structure).  
                  -> menuRscID               Resource ID of the menu.

**Result** Returns nothing.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## FrmSetObjectBounds

**Purpose** Set the bounds or position of an object.

**Declared In** Form.h

**Prototype** void FrmSetObjectBounds (FormType \*formP,  
                  UInt16 objIndex, const RectangleType \*bounds)

**Parameters** -> formP                   Pointer to the form object ([FormType](#) structure).  
                  -> objIndex               Index of an object in the form. You can obtain  
  this by using [FrmGetObjectIndex](#).  
                  -> bounds                 Window-relative bounds. For the following  
  objects, this sets only the position of the top-left  
   corner: label, bitmap, and Graffiti state  
   indicator.

**Result** Returns nothing.

## Forms

### *Form Functions*

---

- Comments** Doesn't update the display.
- Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## FrmSetObjectPosition

- Purpose** Set the position of an object.
- Declared In** Form.h
- Prototype** void FrmSetObjectPosition (FormType \*formP,  
UInt16 objIndex, Coord x, Coord y)
- Parameters**
- |             |  |
|-------------|--|
| -> formP    | Pointer to the form object ( <a href="#">FormType</a> structure).                                |
| -> objIndex | Index of an object in the form. You can obtain this by using <a href="#">FrmGetObjectIndex</a> . |
| -> x        | Window-relative horizontal coordinate.   |
| -> y        | Window-relative vertical coordinate.   |
- Result** Returns nothing.
- See Also** [FrmGetObjectPosition](#), [FrmGetObjectBounds](#)

## FrmSetTitle

- Purpose** Set the title of a form. If the form is visible, draw the new title.
- Declared In** Form.h
- Prototype** void FrmSetTitle (FormType \*formP,  
Char \*newTitle)
- Parameters**
- |          |   |
|----------|---|
| -> formP | Pointer to the form object ( <a href="#">FormType</a> structure). |
|----------|---|

-> newTitle      Pointer to the new title string.

**Result**      Returns nothing.

**Comments**      This function draws the title if the form is visible.

This function saves the pointer passed in newTitle; it does **not** make a copy. The value of newTitle must not be a pointer to a stack-based object.

**Compatibility**      Earlier versions of this function redrew the title without first erasing the old one. This problem was corrected in version 3.0 of Palm OS.

**See Also**      [FrmGetTitle](#), [FrmCopyTitle](#), [FrmCopyLabel](#)

## FrmShowObject

**Purpose**      Set a form object as usable. If the form is visible, draw the object.

**Declared In**      Form.h

**Prototype**      `void FrmShowObject (FormType *formP,  
                  UInt16 objIndex)`

**Parameters**      -> formP      Pointer to the form object ([FormType](#) structure).

-> objIndex      Index of an object in the form. You can obtain this by using [FrmGetObjectIndex](#).

**Result**      Returns nothing.

**Compatibility**      On versions of Palm OS prior to 3.5 this function doesn't affect lists or tables. On Palm OS 3.5 it operates correctly on lists but doesn't have any effect on tables. On Palm OS 4.0 it operates correctly on both lists and tables.

## Forms

### Form Functions

---

If [3.5 New Feature Set](#) is present and the object is an extended gadget, this function calls the gadget's callback with `formGadgetDrawCmd`. See [FormGadgetHandlerType](#).

**See Also** [FrmHideObject](#)

## FrmUpdateForm

**Purpose** Send a [frmUpdateEvent](#) to the specified form.

**Declared In** Form.h

**Prototype** void FrmUpdateForm (UInt16 formId,  
                  UInt16 updateCode)

**Parameters** -> formId      Resource ID of form to update.  
              -> updateCode     An application-defined code that can be used to indicate what needs to be updated. Specify the code `frmRedrawUpdateCode` to indicate that the whole form should be redrawn.

**Result** Returns nothing.

**Comments** If the `frmUpdateEvent` posted by this function is handled by the default form event handler, [FrmHandleEvent](#), the `updateCode` parameter is ignored. `FrmHandleEvent` always redraws the form. If you handle the `frmUpdateEvent` in a custom event handler, you can use the `updateCode` parameter any way you want. For example, you might use it to indicate that only a certain part of the form needs to be redrawn. If you do handle the `frmUpdateEvent`, be sure to return `true` from your event handler so that the default form handler does not also redraw the whole form.

If you do handle the `frmUpdateEvent` in a custom event handler, be sure to handle the case where `updateCode` is set to `frmRedrawUpdateCode`, and redraw the whole form. This event (and code) is sent by the system when the whole form needs to be redrawn because the display needs to be refreshed.

## FrmUpdateScrollers

**Purpose** Visually update (show or hide) the field scroll arrow buttons.

**Declared In** Form.h

**Prototype** void FrmUpdateScrollers (FormType \*formP,  
                  UInt16 upIndex, UInt16 downIndex,  
                  Boolean scrollableUp, Boolean scrollableDown)

**Parameters**

-> formP	Pointer to the form object ( <a href="#">FormType</a> structure).
-> upIndex	Index of the up-scroller button. You can obtain this by using <a href="#">FrmGetObjectIndex</a> .
-> downIndex	Index of the down-scroller button. You can obtain this by using <a href="#">FrmGetObjectIndex</a> .
-> scrollableUp	Set to true to make the up scroll arrow active (shown), or false to hide it.
-> scrollableDown	Set to true to make the down scroll arrow active (shown), or false to hide it.

**Result** Returns nothing.

## FrmValidatePtr

**Purpose** Return true if the specified pointer references a valid form.

**Declared In** Form.h

**Prototype** Boolean FrmValidatePtr (const FormType \*formP)

**Parameters**

-> formP	Pointer to be tested.
----------	-----------------------

**Result** Returns true if the specified pointer is a non-NULL pointer to an object having a valid form structure.

## **Forms**

### *Form Functions*

---

**Comments** This function is intended for debugging purposes only. Do not include it in released code.

To distinguish between a window and a form in released code, instead of using this function, look at the flag `windowFlags.dialog` in the [WindowType](#) structure. This flag is true if the window is a form.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

## FrmVisible

**Purpose** Return `true` if the form is visible (is drawn).

**Declared In** Form.h

**Prototype** Boolean FrmVisible (const FormType \*formP)

**Parameters** -> formP Pointer to the form object ([FormType](#) structure).

**Result** Returns `true` if the form is visible; `false` if it is not visible.

**See Also** [FrmDrawForm](#), [FrmEraseForm](#)

# Application-Defined Functions

## FormCheckResponseFuncType

**Purpose** Callback function for [FrmCustomResponseAlert](#).

**Declared In** Form.h

**Prototype** Boolean FormCheckResponseFuncType (Int16 button,  
Char \*attempt)

**Parameters** -> button The ID of the button that the user tapped.

## Forms

### *Application-Defined Functions*

---

	-> attempt	The string that the user entered in the alert dialog.
<b>Result</b>		Return <code>true</code> if the dialog should be dismissed. Return <code>false</code> if the dialog should not be dismissed.
<b>Comments</b>		<p>This function is called at these times during the <code>FrmCustomResponseAlert</code> routine:</p> <ul style="list-style-type: none"><li>• At the beginning of <code>FrmCustomResponseAlert</code>, this function is called with a button ID of <code>frmResponseCreate</code>. This constant indicates that the dialog is about to be displayed, and your function should perform any necessary initialization. For example, on a Japanese system, a password dialog might need to disable the Japanese FEP. So it would call <code>TsmSetFepMode (NULL, tsmFepModeOff)</code> in this function.</li><li>• When the user has tapped a button on the dialog. The function should process the <code>attempt</code> string. If the string is valid input, the function should return <code>true</code>. If not, it should return <code>false</code> to give the user a chance to re-enter the string.</li><li>• At the end of <code>FrmCustomResponseAlert</code>, this function is called with a button ID of <code>frmResponseQuit</code>. This gives the callback a chance to perform any cleanup, such as re-enabling the Japanese FEP.</li></ul>
<b>Compatibility</b>		Implemented only if <a href="#">3.5 New Feature Set</a> is present.

## FormEventHandlerType

<b>Purpose</b>	The event handler callback routine for a form.
<b>Declared In</b>	Form.h
<b>Prototype</b>	Boolean FormEventHandlerType (EventType *eventP)
<b>Parameters</b>	-> eventP      Pointer to the form event ( <a href="#">FormType</a> structure).
<b>Result</b>	Must return true if this routine handled the event, otherwise false.
<b>Comments</b>	<p><a href="#">FrmDispatchEvent</a> calls this handler whenever it receives an event for the form.</p> <p>This callback routine is the mechanism for dispatching events to particular forms in an application. The callback is registered by the routine <a href="#">FrmSetEventHandler</a>.</p>

## FormGadgetHandlerType

<b>Purpose</b>	The event handler callback for an extended gadget.
<b>Declared In</b>	Form.h
<b>Prototype</b>	Boolean (FormGadgetHandlerType) (struct FormGadgetTypeInCallback *gadgetP, UInt16 cmd, void *paramP)
<b>Parameters</b>	-> gadgetP      Pointer to the gadget structure. See <a href="#">FormGadgetType</a> . -> cmd      A constant that specifies what action the handler should take. This can be one of the following: formGadgetDeleteCmd Sent by <a href="#">FrmDeleteForm</a> to indicate that

## Forms

### *Application-Defined Functions*

---

the gadget is being deleted and must clean up any memory it has allocated or perform other cleanup tasks.

`formGadgetDrawCmd`

Sent by [FrmDrawForm](#) and [FrmShowObject](#) to indicate that the gadget must be drawn or redrawn.

`formGadgetEraseCmd`

Sent by [FrmHideObject](#) to indicate that the gadget is going to be erased. [FrmHideObject](#) clears the visible and usable flags for you. If you return `false`, it also calls [WinEraseRectangle](#) to erase the gadget's bounds.

`formGadgetHandleEventCmd`

Sent by [FrmHandleEvent](#) to indicate that a gadget event has been received. The `paramP` parameter contains the pointer to the `EventType` structure.

`-> paramP`

NULL except if cmd is `formGadgetHandleEventCmd`. In that case, this parameter holds the pointer to the `EventType` structure containing the event.

**Result** Return `true` if the event was handled successfully; `false` otherwise.

**Comments** If this function performs any drawing in response to the `formGadgetDrawCmd`, it should set the gadget's `visible` attribute flag. (`gadgetP->attr.visible = true`). This flag indicates that the gadget appears on the screen. If you don't set the `visible` flag, the gadget won't be erased when [FrmHideObject](#) is called. ([FrmHideObject](#) immediately returns if the object's `visible` flag is `false`.)

Note that if the function receives the `formGadgetEraseCmd`, it may simply choose to perform any necessary cleanup and return `false`. If the function returns `false`, [FrmHideObject](#) erases the

gadget's bounding rectangle. If the function returns `true`, it must erase the gadget area itself.

If this function receives a `formGadgetHandleEventCmd`, `paramP` points one of two events: [frmGadgetEnterEvent](#) or [frmGadgetMiscEvent](#). The `frmGadgetEnterEvent` is passed when there is a [penDownEvent](#) within the gadget's bounds. This function should track the pen and perform any necessary highlighting. The `frmGadgetMiscEvent` is never sent by the system. Your application may choose to use it if at any point it needs to send data to the extended gadget. In this case, the event has one or both of these fields defined: `selector`, an unsigned integer, and `dataP`, a pointer to data.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [FrmSetGadgetHandler](#)

## **Forms**

### *Application-Defined Functions*

---

# 12

## Graffiti Shift

---

This chapter provides reference material for the Graffiti® Shift facility, declared in the header file `GraffitiShift.h`.

### GraffitiShift Functions

#### **GsiEnable**

**Purpose** Enable or disable the Graffiti-shift state indicator.

**Declared In** `GraffitiShift.h`

**Prototype** `void GsiEnable (const Boolean enableIt)`

**Parameters** `enableIt` true to enable, false to disable.

**Result** Returns nothing.

**Comments** Enabling the indicator makes it visible, disabling it makes the insertion point invisible.

## GsiEnabled

**Purpose** Return `true` if the Graffiti-shift state indicator is enabled, or `false` if it's disabled.

**Declared In** `GraffitiShift.h`

**Prototype** `Boolean GsiEnabled (void)`

**Parameters** None.

**Result** `true` if enabled, `false` if not.

## GsiInitialize

**Purpose** Initialize the global variables used to manage the Graffiti-shift state indicator.

**Declared In** `GraffitiShift.h`

**Prototype** `void GsiInitialize (void)`

**Parameters** None.

**Result** Returns nothing.

## GsiSetLocation

**Purpose** Set the display-relative position of the Graffiti-shift state indicator.

**Declared In** `GraffitiShift.h`

**Prototype** `void GsiSetLocation (const Int16 x, const Int16 y)`

**Parameters** `x, y` Coordinate of left side and top of the indicator.

**Result** Returns nothing.

**Comments** The indicator is not redrawn by this routine.

## GsiSetShiftState

**Purpose** Set the Graffiti-shift state indicator.

**Declared In** GraffitiShift.h

**Prototype** void GsiSetShiftState (const UInt16 lockFlags,  
const UInt16 tempShift)

**Parameters** lockFlags        glfCapsLock or glfNumLock.  
tempShift            The current temporary shift.

**Result** Returns nothing.

**Comments** This function affects only the state of the UI element, not the underlying Graffiti engine.

**See Also** [GrfSetState](#)

## **Graffiti Shift**

### *GraffitiShift Functions*

---

# Insertion Point

---

This chapter provides reference material for the insertion point API, declared in the header file `InsPoint.h`.

For more information on the insertion point, see the section “[Insertion Point](#)” in the *Palm OS Programmer’s Companion*, vol. I.

## Insertion Point Functions

### **InsPtEnable**

**Purpose** Enable or disable the insertion point. When the insertion point is disabled, it’s invisible; when it’s enabled, it blinks.

**Declared In** `InsPoint.h`

**Prototype** `void InsPtEnable (Boolean enableIt)`

**Parameters** `enableIt`      `true = enable; false = disable`

**Result** Returns nothing.

**Comments** This function is called by the Form functions when a text field loses or gains the focus, and by the Windows function when a region of the display is copied ([WinCopyRectangle](#)).

**See Also** [InsPtEnabled](#)

## **Insertion Point**

### *Insertion Point Functions*

---

#### **InsPtEnabled**

<b>Purpose</b>	Return <code>true</code> if the insertion point is enabled or <code>false</code> if the insertion point is disabled.
<b>Declared In</b>	<code>InsPoint.h</code>
<b>Prototype</b>	<code>Boolean InsPtEnabled (void)</code>
<b>Parameters</b>	None.
<b>Result</b>	Returns <code>true</code> if the insertion point is enabled (blinking); returns <code>false</code> if the insertion point is disabled (invisible).
<b>See Also</b>	<a href="#">InsPtEnable</a>

#### **InsPtGetHeight**

<b>Purpose</b>	Return the height of the insertion point.
<b>Declared In</b>	<code>InsPoint.h</code>
<b>Prototype</b>	<code>Int16 InsPtGetHeight (void)</code>
<b>Parameters</b>	None.
<b>Result</b>	Returns the height of the insertion point, in pixels.

## InsPtGetLocation

**Purpose** Return the screen-relative position of the insertion point.

**Declared In** InsPoint.h

**Prototype** void InsPtGetLocation (Int16 \*x, Int16 \*y)

**Parameters** x, y Pointer to top-left position of insertion point's x and y coordinate.

**Result** Returns nothing. Stores the location in x and y.

**Comments** This function is called by the Field functions. An application would not normally call this function.

## InsPtSetHeight

**Purpose** Set the height of the insertion point.

**Declared In** InsPoint.h

**Prototype** void InsPtSetHeight (const Int16 height)

**Parameters** height Height of the insertion point in pixels.

**Result** Returns nothing.

**Comments** Set the height of the insertion point to match the character height of the font used in the field that the insertion point is in. When the current font is changed, the insertion point height should be set to the line height of the new font.

If the insertion point is visible when its height is changed, it's erased and redrawn with its new height.

**See Also** [InsPtGetHeight](#)

## **Insertion Point**

### *Insertion Point Functions*

---

## **InsPtSetLocation**

**Purpose** Set the screen-relative position of the insertion point.

**Declared In** InsPoint.h

**Prototype** void InsPtSetLocation (const Int16 x,  
const Int16 y)

**Parameters** x, y Number of pixels from the left side (top) of the display.

**Result** Returns nothing.

**Comments** The position passed to this function is the location of the top-left corner of the insertion point.

This function should be called only by the Field functions.

**See Also** [InsPtGetLocation](#)

# Lists

---

This chapter provides information about list objects by discussing these topics:

- [List Data Structures](#)
- [List Resources](#)
- [List Functions](#)
- [Application-Defined Function](#)

The header file `List.h` declares the API that this chapter describes. For more information on lists, see the section “[Lists](#)” in the *Palm OS Programmer’s Companion*, vol. I.

## List Data Structures

### ListAttrType

The `ListAttrType` bit field defines the visible characteristics of the list.

```
typedef struct {
    UInt16 usable      :1;
    UInt16 enabled     :1;
    UInt16 visible     :1;
    UInt16 poppedUp   :1;
    UInt16 hasScrollBar :1;
    UInt16 search      :1;
    UInt16 reserved    :2;
} ListAttrType;
```

## Lists

### *List Data Structures*

---

#### Field Descriptions

usable	Set to make the list usable.
	If not set, the list is not considered part of the current interface of the application, and does not appear on screen.
enabled	Not used.
visible	Set when the list object is drawn, and cleared when the list object is erased.
	This attribute is set and cleared internally.
poppedUp	Set to indicate that the choices are displayed in a popup window.
	This attribute is set and cleared internally.
hasScrollBar	Set to indicate that the list has a scroll bar.
search	Set to enable incremental search. If incremental search is enabled, when the list is displayed the user can navigate the list by entering up to five characters. The list will scroll to present the first list item that matches the entered characters. This feature only works for popup lists, and only works if the list is sorted and the list items are available to the List Manager (that is, you don't pass NULL to <a href="#">LstSetListChoices</a> ).
reserved	Reserved for system use.

## ListType

The ListType structure is defined below.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the ListType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    UInt16           id;
    RectangleType    bounds;
    ListAttrType     attr;
    Char             **itemsText;
    Int16            numItems;
    Int16            currentItem;
    Int16            topItem;
    FontID           font;
    UInt8            reserved;
    WinHandle        popupWin;
    ListDrawDataFuncPtr drawItemCallback;
} ListType;
```

## Field Descriptions

<b>id</b>	The ID value, specified by the application developer. This ID value is part of the event data of <a href="#">_lstEnterEvent</a> and <a href="#">_lstSelectEvent</a> .
<b>bounds</b>	The bounds of the list, relative to the window. For example, to access the bounds of an object in a form whose ID is kObjectID: { RectangleType rect; FormPtr formP = FrmGetActiveForm();  FrmGetObjectBounds(formP, FrmGetObjectIndex(formP, kObjectID), &rect); }
<b>attr</b>	The list attributes. See <a href="#">ListAttrType</a> .

## Lists

### List Data Structures

---

itemsText

A pointer to an array of pointers to the text of the choices. Access with [LstGetSelectionText](#).

For example, to access the string specified by itemNum in the list whose ID is kChoiceList use the following:

```
{  
    Char *string;  
    Int16 itemNum;  
    ...  
    string =  
        LstGetSelectionText (GetObjectPtr (kChoicesList), itemNum);  
}
```

where GetObjectPtr is the following:

```
static void *GetObjectPtr (UInt16 rsrcID) {  
    FormPtr formP;  
  
    formP = FrmGetActiveForm ();  
    return FrmGetObjectPtr (formP,  
        FrmGetObjectIndex (formP,  
            rsrcID));  
}
```

If you use a callback routine to draw the list items, note that the itemsText pointer you supply to [LstSetListChoices](#) is passed to your callback routine. See the comments under [ListDrawDataFuncType](#) for tips on using itemsText with a callback routine.

numItems

The number of choices in the list. Access with [LstGetNumberOfItems](#).

currentItem

The currently-selected list choice (0 = first choice). Access with [LstGetSelection](#).

topItem

The first choice displayed in the list. Set this field with [LstSetTopItem](#).

font	The ID of the font used to draw all list text strings.
reserved	Reserved for future use.
popupWin	The handle of the window created when a list is displayed by calling <a href="#">LstPopupList</a> .
drawItemCallback	Function used to draw an item in the list. If NULL, the default drawing routine is used instead. Set this field with <a href="#">LstSetDrawFunction</a> . See <a href="#">Application-Defined Function</a> .

## List Resources

The [List Resource](#) (tLST), and [Popup Trigger Resource](#) (tPUT) are used together to represent an active list.

## List Functions

### LstDrawList

**Purpose** Sets the visible attribute of the list object to true, and draws the list object if it is usable.

**Declared In** List.h

**Prototype** void LstDrawList (ListType \*listP)

**Parameters** -> listP Pointer to a list object ([ListType](#)).

**Result** Returns nothing.

**Comments** If there are more choices than can be displayed, this function ensures that the current selection is visible. The current selection is highlighted. Note that this function does not ensure the current

## Lists

### List Functions

---

selection is visible; if you need to do this, call the [LstMakeItemVisible](#) function.

If the list is disabled, it's drawn grayed-out (strongly discouraged). If it's empty, nothing is drawn. If it's not usable, nothing is drawn.

**See Also** [FrmGetObjectPtr](#), [LstPopupList](#), [LstEraseList](#)

## LstEraseList

**Purpose** Erase a list object.

**Declared In** List.h

**Prototype** void LstEraseList (ListType \*listP)

**Parameters** -> listP Pointer to a list object ([ListType](#)).

**Result** Returns nothing.

**Comments** The visible attribute is set to false by this function.

**See Also** [FrmGetObjectPtr](#), [LstDrawList](#)

## LstGetNumberOfItems

**Purpose** Return the number of items in a list.

**Declared In** List.h

**Prototype** Int16 LstGetNumberOfItems (const ListType \*listP)

**Parameters** -> listP Pointer to a list object ([ListType](#)).

**Result** Returns the number of items in a list.

**See Also** [FrmGetObjectPtr](#), [LstSetListChoices](#)

## LstGetSelection

**Purpose** Return the currently selected choice in the list.

**Declared In** List.h

**Prototype** Int16 LstGetSelection (const ListType \*listP)

**Parameters** -> listP Pointer to a list object.

**Result** Returns the item number of the current list choice. The list choices are numbered sequentially, starting with 0; Returns noListSelection if none of the items are selected.

**See Also** [FrmGetObjectPtr](#), [LstSetListChoices](#), [LstSetSelection](#), [LstGetSelectionText](#)

## LstGetSelectionText

**Purpose** Return a pointer to the text of the specified item in the list, or NULL if no such item exists.

**Declared In** List.h

**Prototype** Char \*LstGetSelectionText (const ListType \*listP, Int16 itemNum)

**Parameters** -> listP Pointer to a list object.

-> itemNum Item to select (0 = first item in list).

**Result** Returns a pointer to the text of the current selection, or NULL if out of bounds.

**Comments** This is a pointer within [ListType](#), not a copy. This function is only usable if you supplied an array of strings and a count to

## Lists

### List Functions

---

[LstSetListChoices](#); if your application uses a callback function that dynamically generates the list text, this function returns NULL.

**See Also** [FrmGetObjectPtr](#)

## LstGetTopItem

**Purpose** Returns the topmost visible item.

**Declared In** List.h

**Prototype** Int16 LstGetTopItem (const ListType \*listP)

**Parameters** -> listP Pointer to a list object.

**Result** Returns the item number of the top item visible.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [LstGetSelection](#), [LstSetTopItem](#)

## LstGetVisibleItems

**Purpose** Return the number of visible items.

**Declared In** List.h

**Prototype** Int16 LstGetVisibleItems (const ListType \*listP)

**Parameters** -> listP Pointer to a list object.

**Result** The number of items visible.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## LstHandleEvent

**Purpose** Handle event in the specified list; the list object must have its **usable** and **visible** attribute set to **true**. This routine handles two type of events, [penDownEvent](#) and [lstEnterEvent](#); see [Comments](#).

**Declared In** List.h

**Prototype** Boolean LstHandleEvent (ListType \*listP,  
const EventType \*eventP)

**Parameters** -> listP Pointer to a list object ([ListType](#)).  
-> eventP Pointer to an EventType structure.

**Result** Return **true** if the event was handled. The following cases will result in a return value of **true**:

- A penDownEvent within the bounds of the list
- A lstEnterEvent with a list ID value that matches the list ID in the list data structure

**Comments** When this routine receives a penDownEvent, it checks if the pen position is within the bounds of the list object. If it is, this routine tracks the pen until the pen comes up. If the pen comes up within the bounds of the list, a lstEnterEvent is added to the event queue, and the routine is exited.

When this routine receives a lstEnterEvent, it checks that the list ID in the event record matches the ID of the specified list. If there is a match, this routine creates and displays a popup window containing the list's choices and the routine is exited.

If a penDownEvent is received while the list's popup window is displayed and the pen position is outside the bounds of the popup window, the window is dismissed. If the pen position is within the bounds of the window, this routine tracks the pen until it comes up. If the pen comes up outside the list object, a lstEnterEvent is added to the event queue.

## Lists

### List Functions

---

## LstMakeItemVisible

**Purpose** Make an item visible, preferably at the top. If the item is already visible, make no changes.

**Declared In** List.h

**Prototype** void LstMakeItemVisible (ListType \*listP,  
Int16 itemNum)

**Parameters** -> listP Pointer to a list object ([ListType](#)).  
-> itemNum Item to select (0 = first item in list).

**Result** Returns nothing.

**Comments** Does *not* visually update the list. You must call [LstDrawList](#) to update it.

**See Also** [FrmGetObjectPtr](#), [LstSetSelection](#), [LstSetTopItem](#),  
[LstDrawList](#)

## LstNewList

**Purpose** Create a new list object dynamically and install it in the specified form. This function can be used to create a new popup trigger and its associated list.

**Declared In** List.h

**Prototype** Err LstNewList (void \*\*formPP, UInt16 id,  
Coord x, Coord y, Coord width, Coord height,  
FontID font, Int16 visibleItems, Int16 triggerId)

<b>Parameters</b>	<-> formPP	Pointer to the pointer to the form in which the new list is installed. This value is not a handle; that is, the old formPP value is not necessarily valid after this function returns. In subsequent calls, always use the new formPP value returned by this function.
	-> id	Symbolic ID of the list, specified by the developer. By convention, this ID should match the resource ID (not mandatory).
	-> x	Horizontal coordinate of the upper-left corner of the list's boundaries, relative to the window in which it appears.
	-> y	Vertical coordinate of the upper-left corner of the list's boundaries, relative to the window in which it appears.
	-> width	Width of the list, expressed in pixels. Valid values are 1 – 160.
	-> height	Height of the list, expressed in pixels. Valid values are 1 – 160.
	-> visibleItems	Number of list items that can be viewed together.

## Lists

### List Functions

---

-> triggerId      Symbolic ID of the popup trigger associated with the new list (this ID is specified by the developer). A nonzero value for triggerID causes this function to create both the list and its associated popup trigger. If the list isn't a popup list, pass 0 for triggerId.

**Result**      Returns 0 if no error.

**Compatibility**      Implemented only if [3.0 New Feature Set](#) is present.

**See Also**      [LstDrawList](#), [FrmRemoveObject](#)

## LstPopupList

**Purpose**      Display a modal window that contains the items in the list.

**Declared In**      List.h

**Prototype**      Int16 LstPopupList (ListType \*listP)

**Parameters**      -> listP      Pointer to a list object.

**Result**      Returns the list item selected, or -1 if no item was selected.

**Comments**      Saves the previously active window. Creates and deletes the new popup window.

**See Also**      [FrmGetObjectPtr](#)

## LstScrollList

**Purpose** Scroll the list up or down a number of times.

**Declared In** List.h

**Prototype** Boolean LstScrollList (ListType \*listP,  
WinDirectionType direction, Int16 itemCount)

**Parameters**

-> listP	Pointer to a list object.
-> direction	Direction to scroll.
-> itemCount	Items to scroll in direction.

**Result** Returns true when the list is actually scrolled, false otherwise.  
May return false if a scroll past the end of the list is requested.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## LstSetDrawFunction

**Purpose** Set a callback function to draw each item instead of drawing the item's text string.

**Declared In** List.h

**Prototype** void LstSetDrawFunction (ListType \*listP,  
ListDrawDataFuncPtr func)

**Parameters**

-> listP	Pointer to a list object.
-> func	Pointer to a function that draws items.

**Result** Returns nothing.

## Lists

### List Functions

---

**Comments** This function also adjusts `topItem` to prevent a shrunken list from being scrolled down too far. Use this function for custom draw functionality.

**See Also** [FrmGetObjectPtr](#), [LstSetListChoices](#),  
[ListDrawDataFuncType](#)

## LstSetHeight

**Purpose** Set the number of items visible in a list.

**Declared In** List.h

**Prototype** void LstSetHeight (ListType \*listP,  
Int16 visibleItems)

**Parameters** -> listP Pointer to a list object.

-> visibleItems Number of choices visible at once.

**Result** Returns nothing.

**Comments** This function doesn't redraw the list if it's already visible.

**See Also** [FrmGetObjectPtr](#)

## LstSetListChoices

**Purpose** Set the items of a list to the array of text string pointers passed to this function. This functions erases the old list items.

**Declared In** List.h

**Prototype** void LstSetListChoices (ListType \*listP,  
Char \*\*itemsText, Int16 numItems)

**Parameters** -> listP Pointer to a list object.

-> itemsText      Pointer to an array of text strings. See  
[SysFormPointerArrayToStrings](#) for one  
way to create this array of strings.  
-> numItems      Number of choices in the list.

**Result** Returns nothing.

**Comments** You need to call the `LstDrawList` function to update the list if it is displayed when you call this function.

---

**NOTE:** This function does not copy the strings in the `itemsText` array, which means that you need to ensure that the array is not moved or deallocated until after you are done with the list.

---

If you use a callback routine to draw the items in your list, the `itemsText` pointer is simply passed to that callback routine and is not otherwise used by the List Manager code. See the comments under [ListDrawDataFuncType](#) for tips on using the `itemsText` parameter with a callback routine.

**See Also** [FrmGetObjectPtr](#), [LstSetSelection](#), [LstSetTopItem](#),  
[LstDrawList](#), [LstSetHeight](#), [LstSetDrawFunction](#)

## LstsetPosition

**Purpose** Set the position of a list.

**Declared In** List.h

**Prototype** void LstsetPosition (ListType \*listP, Coord x,  
Coord y)

**Parameters** -> listP      Pointer to a list object  
-> x      Left bound.

## Lists

### List Functions

---

- > Y Top bound.

**Result** Returns nothing.

**Comments** The list is not redrawn. Don't call this function when the list is visible.

**See Also** [FrmGetObjectPtr](#)

## LstSetSelection

**Purpose** Set the selection for a list.

**Declared In** List.h

**Prototype** void LstSetSelection (ListType \*listP,  
Int16 itemNum)

**Parameters** - > listP Pointer to a list object.

- > itemNum Item to select (0 = first item in list,  
noListSelection = none).

**Result** Returns nothing.

**Comments** The old selection, if any, is unselected. If the list is visible, the selected item is visually updated. The list is scrolled to the selection, if necessary, as long as the list object is both visible and usable.

**See Also** [FrmGetObjectPtr](#), [LstSetTopItem](#)

## LstSetTopItem

**Purpose** Set the item visible. The item cannot become the top item if it's on the last page.

**Declared In** List.h

**Prototype** void LstSetTopItem (ListType \*listP,  
Int16 itemNum)

**Parameters** -> listP Pointer to a list object.  
-> itemNum Item to select (0 = first item in list). This must be a valid item number.

**Result** Returns nothing.

**Comments** Does *not* update the display.

---

**NOTE:** The value you specify for itemNum must be in the range 0 to max-number-of-items.

---

**See Also** [FrmGetObjectPtr](#), [LstSetSelection](#), [LstGetTopItem](#),  
[LstMakeItemVisible](#), [LstDrawList](#), [LstEraseList](#)

## Application-Defined Function

If you need to perform special drawing for items in the list, call [LstSetDrawFunction](#) to set the list drawing callback function. The [ListDrawDataFuncType](#) section documents the prototype for the callback function you provide for drawing list items.

### ListDrawDataFuncType

**Purpose** Callback function that you provide for drawing items in a list. This function is called whenever the Palm OS needs to draw an element

## Lists

### *Application-Defined Function*

---

in the list. Your callback function declaration must match the prototype shown here.

**Declared In** List.h

**Prototype** void ListDrawDataFuncType(Int16 itemNum,  
RectangleType \*bounds, Char \*\*itemsText)

**Parameters**

-> itemNum	The number of the item to draw.
-> bounds	The bounds of the list, relative to the window.
-> itemsText	A pointer to an array of pointers to the text of the list items. This is the pointer that you supplied when calling <a href="#">LstSetListChoices</a> .

**Result** Returns nothing.

**Comments** You can call [LstSetDrawFunction](#) to register your callback function for the list, which means that your function will be called to draw the list items, rather than using the built-in draw functionality, which displays each item's text string.

Your callback function is called whenever an item in the list needs to be drawn. When it is called, the value of the itemNum parameter specifies which item the function is to draw. The itemsText parameter, which is what was supplied to LstSetListChoices, doesn't actually need to point to a list of strings: you can pass NULL, or you can pass a pointer to anything that will be useful to your drawing function. Note, however, that if you pass anything other than a pointer to a list of strings when you call LstSetListChoices, you must ensure that [LstGetSelectionText](#) is never called since it assumes that this pointer indicates an array of text items. In particular, if your list is associated with a pop-up trigger you must handle the popSelectEvent yourself before [FrmHandleEvent](#) gets a chance at it since FrmHandleEvent calls LstGetSelectionText.

**WARNING!** If the list is a popup list, the form that owns the list is not active while the list is in a window. This means that you cannot call the [FrmGetActiveForm](#) function. Instead, use itemsText pointer to access any information that you need for drawing. If you must access the form, use the [FrmGetFormPtr](#) function.

---

Note that the list object manages which colors are used to draw the items and how to draw selected versus unselected items. In almost all circumstances, your drawing function does not have to be concerned with these details.

However, if you do need to determine if the item is selected, you can rely on the fact that the system has set the pen color to one of two colors prior to calling your draw function:

- If the item is selected, the foreground color is `UIObjectSelectedForeground`.
- If the item is not selected, the foreground color is `UIObjectForeground`.

You can determine the foreground color that is in effect for the list item by calling the [WinSetForeColor](#) function, which returns the previous value of the foreground color. Remember to call [WinSetForeColor](#) again to reset the foreground color to what it was. For example:

```
itemColor = WinSetForeColor(myColor)
WinSetForeColor(itemColor)
selectColor = UIColorGetTableEntryIndex(
                UIObjectSelectedForeground)
if itemColor == selectColor
    ...

```

**See Also** [LstSetDrawFunction](#), [UIColorGetTableEntryIndex](#), [WinSetForeColor](#)

## **Lists**

*Application-Defined Function*

---

# Menus

---

This chapter describes the menu API as declared in the header file `Menu.h`. It discusses the following topics:

- [Menu Data Structures](#)
- [Menu Constants](#)
- [Menu Resources](#)
- [Menu Functions](#)

For more information on menus, see the section “[Menus](#)” on page 105 in the *Palm OS Programmer’s Companion*, vol. I.

## Menu Data Structures

### MenuBarAttrType

The `MenuBarAttrType` bit field defines some characteristics of the menu bar.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `MenuBarAttrType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    UInt16 visible : 1;
    UInt16 commandPending : 1;
    UInt16 insPtEnabled : 1;
    UInt16 needsRecalc : 1;
} MenuBarAttrType;
```

## Menus

### Menu Data Structures

---

Your code should treat the `MenuBarAttrType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

#### Field Descriptions

<code>visible</code>	If set, the menu bar is drawn and visible on the screen. This attribute is set as part of <a href="#"><code>MenuDrawMenu</code></a> , which is called when the menu is drawn.
<code>commandPending</code>	If set, a menu command shortcut is in progress. This bit is set during <a href="#"><code>MenuHandleEvent</code></a> if the menu shortcut keystroke is received. If the next key is received before the timeout value is reached, the key is examined to see if it is a valid menu command.
<code>insPtEnabled</code>	Stores the state of the insertion point at the time the menu was drawn so that it can be restored when the menu is erased.
<code>needsRecalc</code>	If set, recalculate menu dimensions.

#### Compatibility

The `needsRecalc` constant is present only if [3.5 New Feature Set](#) is present.

## MenuCmdBarButtonType

The `MenuCmdBarButtonType` structure defines a button to be displayed on the command toolbar. The `buttonsData` field of the [`MenuCmdBarType`](#) structure contains an array of structures of this type.

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `MenuCmdBarButtonType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    UInt16    bitmapId;
    Char      name [menuCmdBarMaxTextLength];
    MenuCmdBarResultType resultType;
    UInt8     reserved;
    UInt32    result;
} MenuCmdBarButtonType;
```

Your code should treat the `MenuCmdBarButtonType` structure as opaque. Do not attempt to change structure member values directly. Instead, use [MenuCmdBarAddButton](#) to add a button to the display. For the most part, the parameters to `MenuCmdBarAddButton` are the same as the fields in the `MenuCmdBarButtonType`, so there should be no need to alter these fields directly.

[MenuCmdBarGetButtonData](#) can be called to access information about command bar buttons.

### Field Descriptions

<code>bitmapId</code>	Resource ID of the bitmap to display on the button. This bitmap should be 13 pixels high by 16 pixels wide.
<code>name</code>	Text to display in the status message when the user taps the button.
<code>resultType</code>	Specifies what type of data is contained in the <code>result</code> field. See <a href="#">MenuCmdBarResultType</a> .

## Menus

### Menu Data Structures

---

reserved	Reserved for future use.
result	Specifies the data to send when the user clicks the button. The data is interpreted as specified by the resultType field. The result can be a shortcut character to enqueue in a <a href="#">keyDownEvent</a> , a menu item ID to enqueue in a <a href="#">menuEvent</a> , or a notification to be broadcast.

**Compatibility** This structure is defined only if [3.5 New Feature Set](#) is present.

## MenuCmdBarResultType

The MenuCmdBarResultType enum specifies how the result field in the [MenuCmdBarButtonType](#) structure should be interpreted.

```
typedef enum {
    menuCmdBarResultNone,
    menuCmdBarResultChar,
    menuCmdBarResultMenuItem,
    menuCmdBarResultNotify
} MenuCmdBarResultType;
```

### Value Descriptions

menuCmdBarResultNone	Send nothing.
menuCmdBarResultChar	The result is a character to send in a <a href="#">keyDownEvent</a> .
menuCmdBarResultMenuItem	The result is the ID of the menu item to send in a <a href="#">menuEvent</a> .
menuCmdBarResultNotify	The result is a notification constant to be broadcast using <a href="#">SysNotifyBroadcastDeferred</a>

**Compatibility** This enum is defined only if [3.5 New Feature Set](#) is present.

## MenuCmdBarType

The MenuCmdBarType structure defines the command toolbar. This command toolbar is allocated and displayed when the user draws the shortcut stroke in the Graffiti® area. It is deallocated when [MenuEraseStatus](#) is called, which occurs most frequently when the timeout value has elapsed.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the MenuCmdBarType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct MenuCmdBarType {  
    WinHandle    bitsBehind;  
    Int32        timeoutTick;  
    Coord        top;  
    Int16        numButtons;  
    Boolean      insPtWasEnabled;  
    Boolean      gsiWasEnabled;  
    Boolean      feedbackMode;  
    MenuCmdBarButtonType *buttonsData;  
} MenuCmdBarType;
```

Your code should treat the MenuCmdBarType structure as opaque. Do not attempt to change structure member values directly.

### Field Descriptions

**bitsBehind** Handle for the window that contains the region obscured by the command toolbar.

**timeoutTick** Timeout value given in system ticks. If the user hasn't specified a command after this many ticks, the command toolbar is erased from the screen and deallocated from memory. This value also specifies how long the status message is displayed after the user successfully enters a command.

## Menus

### Menu Data Structures

---

top	The top bound of the command toolbar given in display-relative coordinates. The command toolbar is as wide as the screen and displays at the bottom of the screen.
numButtons	Number of buttons displayed on the command toolbar.
insPtWasEnabled	If true, the insertion point was enabled before the command toolbar was displayed and should be re-enabled when the command toolbar is erased. If false, the insertion point was disabled.
gsiWasEnabled	If true, the Graffiti shift indicator was enabled before the command toolbar was displayed and should be re-enabled when the command toolbar is erased. If false, the Graffiti shift indicator was disabled.
feedbackMode	If true, the command toolbar is currently displaying a status message. The status message is displayed to tell the user what command is being performed. If false, the command toolbar is awaiting input.
buttonsData	The list of buttons to display on the command toolbar. See <a href="#">MenuBarButtonType</a> . Buttons are stored in this list sequentially with the rightmost button at index 0. Access with <a href="#">MenuBarGetButtonData</a> .

**Compatibility** This structure is defined only if [3.5 New Feature Set](#) is present.

## MenuBarPtr

The `MenuBarPtr` type defines a pointer to a [MenuBarType](#).

```
typedefMenuBarType *MenuBarPtr;
```

## MenuBarType

The `MenuBarType` structure defines the menu bar. There is one menu bar per form.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `MenuBarType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    WinHandle      barWin;
    WinHandle      bitsBehind;
    WinHandle      savedActiveWin;
    WinHandle      bitsBehindStatus;
    MenuBarAttrType attr;
    Int16          curMenu;
    Int16          curItem;
    Int32          commandTick;
    Int16          numMenus;
    MenuPullDownPtr menus;
} MenuBarType;
```

Your code should treat the `MenuBarType` structure as opaque. Do not attempt to change structure member values directly.

### Field Descriptions

<code>barWin</code>	Handle for the window that contains the menu bar.
<code>bitsBehind</code>	Handle for the window that contains the region obscured by the menu bar.
<code>savedActiveWin</code>	Handle where the currently active window is saved so that it can be restored when the menu is erased.

## Menus

### Menu Data Structures

---

bitsBehindStatus	Handle where the bits behind the status message are saved so that when the message display terminates, the bits can be restored.
	The status message is displayed when the user activates the menu through the use of the command keystroke.
attr	Menu bar attributes. See <a href="#">MenuBarAttrType</a> .
curMenu	Menu number for the currently visible menu. Menus are numbered sequentially starting with 0. The value is preserved when the menu bar is dismissed. A value of noMenuSelection indicates that there is no current pull-down menu.
curItem	Item number of the currently highlighted menu item. The items in each menu are numbered sequentially, starting with zero.
	A value of noMenuItemSelection indicates that there is no current item selected.
commandTick	Tick count at which the status message should be erased.
numMenus	Number of pull-down menus on the menu bar.
menus	Array of <a href="#">MenuPullDownType</a> structures.

### Compatibility

If [3.5 New Feature Set](#) is present, the bitsBehindStatus and commandTick fields are defined but are not used. Instead, the bitsBehind and timeoutTick fields in [MenuCmdBarType](#) define the save-behind window and the timeout value for the command toolbar.

## **MenuItemType**

The MenuItemType structure defines a specific item within a menu. The items array in the [MenuPullDownType](#) structure contains one MenuItemType structure for each menu item in the pull-down menu.

If [3.5 New Feature Set](#) is present, you can add a menu item to a pull-down menu programmatically using [MenuAddItem](#).

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the MenuItemType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

## Menus

### Menu Data Structures

---

```
typedef struct {
    UInt16    id;
    Char      command;
    UInt8     hidden: 1;
    UInt8     reserved: 7;
    Char      *itemStr;
} MenuItemType;
```

### Field Descriptions

id	ID value you specified when you created the menu item. This ID value is included as part of the event data of a <a href="#">menuEvent</a> .
command	Shortcut key. If you provide shortcuts, make sure that each shortcut is unique among all commands available at that time.
hidden	If <code>true</code> , the menu item is hidden. If <code>false</code> , it is displayed. You can set and clear this value using <a href="#">MenuHideItem</a> and <a href="#">MenuShowItem</a> .
reserved	Reserved for future use.
itemStr	Pointer to the text to display for this menu item, including the shortcut key. To include a shortcut key, begin the string with the item's text, then type a tab character, and then the item's shortcut key.  To create a separator bar, create a one-character string containing the <code>MenuSeparatorChar</code> constant.

### Compatibility

The `hidden` and `reserved` fields are defined only if [3.5 New Feature Set](#) is present.

## MenuPullDownPtr

The MenuPullDownPtr type defines a pointer to a [MenuPullDownType](#).

```
typedef MenuPullDownType *MenuPullDownPtr;
```

## MenuPullDownType

The MenuPullDownType structure defines a specific menu accessed from the menu bar. The menus array in the [MenuBarType](#) structure contains one MenuPullDownType structure for each pull-down menu associated with the menu bar.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the MenuPullDownType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    WinHandle      menuWin;
    RectangleType bounds;
    WinHandle      bitsBehind;
    RectangleType titleBounds;
    Char           *title;
    UInt16          hidden : 1;
    UInt16          numItems : 15;
    MenuItemType   *items;
} MenuPullDownType;
```

### Field Descriptions

menuWin	Handle for the window that contains the menu.
bounds	Position and size, in pixels, of the pull-down menu.
bitsBehind	Handle of a window that contains the region obscured by the menu.

## Menus

### Menu Constants

---

title	The menu title (null-terminated string) displayed in the menu bar.
titleBounds	Position and size, in pixels, of the title in the menu bar.
hidden	If <code>true</code> , the menu is hidden; if <code>false</code> , it is displayed. This field is not currently used.
numItems	Number of items in the menu. Separators count as items.
items	Array of <a href="#">MenuItemType</a> structures.

**Compatibility** The `hidden` field is defined only if [3.5 New Feature Set](#) is present.

## Menu Constants

---

Constant	Value	Description
noMenuSelection	-1	The <code>curMenu</code> field of <a href="#">MenuBarType</a> is set to this when there is no currently selected menu.
noMenuItemSelection	-1	The <code>curItem</code> field of <a href="#">MenuBarType</a> is set to this when there is no currently selected menu item.
separatorItemSelected	-2	The <code>curItem</code> field of <a href="#">MenuBarType</a> is set to this when a menu separator item is selected.
MenuSeparatorChar	'-'	Special character indicating that the menu item is a bar used to separate groups of related menu items. The first character of the <code>itemStr</code> string in <a href="#">MenuItemType</a> is set to this.

---

## Menu Resources

The menu bar (MBAR) and pull-down menu (MENU) resources are used jointly to represent a menu object on screen. See “[Menus and Menu Bars](#)” in [Chapter 2, “Palm OS Resources.”](#)

## Menu Functions

### MenuAddItem

**Purpose** Adds an item to the currently active menu.

**Declared In** Menu.h

**Prototype** Err MenuAddItem (UInt16 positionId, UInt16 id, Char cmd, const Char \*textP)

<b>Parameters</b>	-> positionId	ID of an existing menu item. The new menu item is added after this menu item.
	-> id	ID value to use for the new menu item.
	-> cmd	Shortcut key. If you provide shortcuts, make sure that each shortcut is unique among all commands available at that time.
	-> textP	Pointer to the text to display for this menu item, including the shortcut key. To include a shortcut key, begin the string with the item’s text, then type a tab character, and then the item’s shortcut key.  To create a separator bar, create a one-character string containing the MenuSeparatorChar constant.

**Result** Returns 0 upon success or one of the following if an error occurs:  
menuErrNoMenu The textP parameter is NULL.

## Menus

### Menu Functions

---

menuErrSameId    The menu already contains an item with the ID id.

menuErrNotFound    The menu doesn't contain an item with the ID positionId.

May display a fatal error message if there is no current menu.

**Comments**    This function creates a new [MenuItemType](#) structure and adds it to the [MenuBarType](#)'s item list.

You should call this function only in response to a [menuOpenEvent](#). This event is generated when the menu is first made active. In general, a form's menu becomes active the first time a [keyDownEvent](#) with a vchrMenu or vchrCommand is generated, and it remains active until a new form (including a modal form or alert panel) is displayed or until [FrmSetMenu](#) is called to change the form's menu. Palm OS® user interface guidelines discourage adding or hiding menu items at any time other than when the menu is first made active.

**Compatibility**    Implemented only if [3.5 New Feature Set](#) is present.

## MenuCmdBarAddButton

**Purpose**    Defines a button to be displayed on the command toolbar.

**Declared In**    Menu.h

**Prototype**    Err MenuCmdBarAddButton (UInt8 where,  
                  UInt16 bitmapId, MenuCmdBarResultType resultType,  
                  UInt32 result, Char \*nameP)

**Parameters**    -> where    Either menuCmdBarOnLeft to add the button to the left of the other buttons on the command toolbar, menuCmdBarOnRight to add it to the right of the other buttons, or a number indicating the exact position of the button. Button positions are numbered from right to left, and the rightmost position is number 1.

-> bitmapId	Resource ID of the bitmap to display on the button. The bitmap's dimensions should be 13 pixels high by 16 pixels wide.
-> resultType	The type of data contained in the <code>result</code> parameter. See <a href="#">MenuCmdBarResultType</a> .
-> result	The data to send when the user taps this button. This can be a character, a menu item ID, or a notification constant.
-> nameP	Pointer to the text to display in the status message if the user taps the button. If NULL, the text is taken from the menu item that matches the ID or shortcut character contained in <code>result</code> , if a match is found.  If you supply a text buffer for this parameter, <code>MenuCmdBarAddButton</code> makes a copy of the buffer.

**Result** Returns 0 upon success, or one of the following error codes:

`menuErrOutOfMemory`

There is not enough memory available to perform the operation.

`menuErrTooManyItems`

The command toolbar already has the maximum number of buttons allowed (currently 8).

**Comments**

Call this function in response to a [menuCmdBarOpenEvent](#) or to the notification [sysNotifyMenuCmdBarOpenEvent](#). Both of these signal that the user has entered the command keystroke and the command toolbar is about to open. Your response should be to add buttons to the toolbar and to return `false`, indicating that you have not completely handled the event.

The `sysNotifyMenuCmdBarOpenEvent` notification is intended to be used only by shared libraries, system extensions, and other code resources that do not use an event loop. If you're writing an application, always respond to the event instead of the notification; an application should only add buttons to the toolbar if it is the

## Menus

### *Menu Functions*

---

current application. If you register for the notification, you receive it each time the command toolbar is displayed, whether your application is active or not.

Note that the command toolbar is allocated each time it is opened and is deallocated when it is erased from the screen.

There is a limited amount of space in which to display buttons on the command toolbar. You should limit the number of buttons to four or five. The maximum allowed by the system is eight, but you should leave space for the status message that appears after the user chooses an action. Buttons should be contextual; for example, the field code only displays a paste button if there is text on the clipboard. Bitmaps for the buttons should be 16 X 13 pixels.

If a field has focus when the command toolbar is opened, the field manager adds buttons for cut, copy, paste, and undo. If your application does not want this default behavior, set the `preventFieldButtons` field in the `menuCmdBarOpenEvent` structure to `true`. (Note that there is no way to prevent the field buttons from being drawn from within a notification handler.)

The following bitmaps for command toolbar buttons are defined in `UIResources.h`. The system and the built-in applications use these bitmaps to represent the commands listed in the table. Your application should also use them if it performs the same actions. If you use any of these buttons, add them in the order shown from right to left. (For example, `BarDeleteBitmap`, if used, should always be the rightmost button.)

---

<b>Bitmap</b>	<b>Command</b>
<code>BarDeleteBitmap</code>	Delete record.
<code>BarPasteBitmap</code>	Paste clipboard contents at insertion point.
<code>BarCopyBitmap</code>	Copy selection.
<code>BarCutBitmap</code>	Cut selection.
<code>BarUndoBitmap</code>	Undo previous action.
<code>BarSecureBitmap</code>	Show Security dialog.

<b>Bitmap</b>	<b>Command</b>
BarBeamBitmap	Beam current record.
BarInfoBitmap	Show Info dialog (Launcher).

It is best to add buttons on the left side. If you add buttons to the right, this function moves all existing buttons over one position to the left. You can also specify an exact position for the *where* parameter. The positions are numbered from right to left with the rightmost position being 1. If you specify an exact position, the function leaves space for the other buttons. For example, if you specify position 3 and there are no buttons displayed at positions 1 and 2, there will be blank spots to the right of your button.

The *result* and *resultType* parameters specify what the result should be if the user taps the button. *result* contains the actual data, and *resultType* contains a constant that specifies the type of data in *result*. Typically, the *result* is to enqueue a [menuEvent](#). In this case, *resultType* is *menuCmdBarResultMenuItem* and the *result* is the ID of the menu item that should included in the event.

You may also specify the shortcut character instead of the menu ID; however, doing so is inefficient. When *result* is a shortcut character, the [MenuHandleEvent](#) function enqueues a [keyDownEvent](#) with the character in *result*. During the next cycle of the event loop, *MenuHandleEvent* enqueues a *menuEvent* in response to the *keyDownEvent*. Thus, it is better to have your button enqueue the *menuEvent* directly.

If you call *MenuCmdBarAddButton* outside of an application, you might not know of any menu items in the active menu (unless your code has added one using [MenuAddItem](#)). In this case, specify a notification to be broadcast. The notification is broadcast at the top of the next event loop, and it must contain no custom data. (Applications may also use the notification result type.)

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [MenuCmdBarDisplay](#), [MenuCmdBarGetButtonData](#)

## Menus

### *Menu Functions*

---

## MenuCmdBarDisplay

<b>Purpose</b>	Displays the command toolbar.
<b>Declared In</b>	Menu.h
<b>Prototype</b>	<code>void MenuCmdBarDisplay (void)</code>
<b>Parameters</b>	None
<b>Result</b>	Returns nothing.
<b>Comments</b>	This function displays the command toolbar when the user enters the command keystroke. You normally do not call this function in your own code. The form manager calls it at the end of its handling of <a href="#">menuCmdBarOpenEvent</a> .
<b>Compatibility</b>	Implemented only if <a href="#">3.5 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">MenuCmdBarAddButton</a> , <a href="#">MenuCmdBarGetButtonData</a>

## MenuCmdBarGetButtonData

<b>Purpose</b>	Gets the data for a given command button.
<b>Declared In</b>	Menu.h
<b>Prototype</b>	<code>Boolean MenuCmdBarGetButtonData (Int16 buttonIndex, UInt16 *bitmapIdP, MenuCmdBarResultType *resultTypeP, UInt32 *resultP, Char *nameP)</code>
<b>Parameters</b>	-> <code>buttonIndex</code> Index of the button for which you want to obtain information. Buttons are ordered from right to left, with the rightmost button at index 0.

<- bitmapIdP	The resource ID of the bitmap displayed on the button. Pass NULL if you don't want to retrieve this value.
<- resultTypeP	The type of action this button takes. Pass NULL if you don't want to retrieve this value.
<- resultP	The result of tapping the button. Pass NULL if you don't want to retrieve this information.
<- nameP	The text displayed in the status message when this button is tapped. Pass NULL if you don't want to retrieve this information. If not NULL, nameP must point to a string of menuCmdBarMaxTextLength size.

**Result** Returns true if the information was retrieved successfully, false if there is no command toolbar or if there is no button at buttonIndex.

**Comments** You can use this function to retrieve information about the buttons that are displayed on the command toolbar. If the command toolbar has not yet been initialized, this function returns false.

Note that the command toolbar is allocated when the user enters the command keystroke and deallocated when [MenuEraseStatus](#) is called. Thus, the only logical place to call MenuCmdBarGetButtonData is in response to a [menuCmdBarOpenEvent](#) or [sysNotifyMenuCmdBarOpenEvent](#) notification.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [MenuCmdBarDisplay](#), [MenuCmdBarAddButton](#)

## Menus

### Menu Functions

---

## MenuDispose

<b>Purpose</b>	Releases any memory allocated to the menu and the command status and restore any saved bits to the screen.
<b>Declared In</b>	Menu.h
<b>Prototype</b>	void MenuDispose (MenuBarType *menuP)
<b>Parameters</b>	-> menuP Pointer to the menu object to dispose. (See <a href="#">MenuBarType</a> .) If NULL, this function returns immediately.
<b>Result</b>	Returns nothing.
<b>Comments</b>	Most applications do not need to call this function directly. MenuDispose is called by the system when the form that contains the menu is no longer the active form, when the form that contains the menu is freed, and when <a href="#">FrmSetMenu</a> is called to change the form's menu bar.
<b>See Also</b>	<a href="#">MenuInit</a> , <a href="#">MenuDrawMenu</a>

## MenuDrawMenu

<b>Purpose</b>	Draws the current menu bar and the last pull-down that was visible.
<b>Declared In</b>	Menu.h
<b>Prototype</b>	void MenuDrawMenu (MenuBarType *menuP)
<b>Parameters</b>	-> menuP Pointer to a <a href="#">MenuBarType</a> . Must not be NULL.
<b>Result</b>	Returns nothing.
<b>Comments</b>	Most applications do not need to call this function directly. <a href="#">MenuHandleEvent</a> calls MenuDrawMenu when the user taps the

Menu silk-screen button (or taps the form's title on Palm OS 3.5 and higher).

The menu bar and the pull-down menu are drawn in front of all the application windows. The state of the insertion point, the bits that are obscured by the menu bar and the pull-down menu, and the currently active window are saved before the menu is drawn. These are all restored when the menu is erased.

A menu keeps track of the last pull-down menu displayed for as long as the menu is active. A menu loses its active status under these conditions:

- When [FrmSetMenu](#) is called to change the active menu on the form.
- When a new form, even a modal form or alert panel, becomes active.

Suppose a user selects your application's About item from the Options menu then clicks the OK button to return to the main form. When the About dialog is displayed, it becomes the active form, which causes the main form's menu state to be erased. This menu state is not restored when the main form becomes active again. The next time [MenuDrawMenu](#) is called (that is, the next time the user taps the Menu silk-screen button), the menu bar is displayed as it was before, and the first pull-down menu listed in the menu bar is displayed instead of the Options pull-down menu.

**See Also** [MenuInit](#), [MenuDispose](#)

## Menus

### *Menu Functions*

---

## MenuEraseStatus

**Purpose** Erases the menu command status.

**Declared In** Menu.h

**Prototype** void MenuEraseStatus (MenuBarType \*menuP)

**Parameters** -> menuP Pointer to a [MenuBarType](#), or NULL for the current menu.

**Result** Returns nothing.

**Comments** When the user selects a menu command using the command keystroke, the command toolbar or status message is displayed at the bottom of the screen. `MenuEraseStatus` erases the toolbar or status message.

Under most circumstances, you do not need to call this function explicitly—just let the current menu command status remove itself automatically. Otherwise, you may cause text to be erased before the user has a chance to see it.

You need to call `MenuEraseStatus` explicitly only if the command toolbar covers something that is going to be changed by the menu command the user has selected. For example, if the user selects a command that displays a new form, call `MenuEraseStatus` before executing the command. Also, if the command performs some drawing in the lower portion of the window, call `MenuEraseStatus` before performing the drawing function.

**Compatibility** The exact behavior when a menu shortcut character is entered depends on which version of the operating system is running. In versions prior to release 3.5, the system displays the string “Command:” in the lower-left portion of the screen when the user enters the Graffiti command keystroke.

In Palm OS 3.5 and higher, entering the Graffiti command keystroke displays the command toolbar. This toolbar is the entire width of the screen and it displays buttons that the user can tap instead of entering another keystroke. If the user taps a button or enters a

character that matches a shortcut character for an item on the active menu, a status message is displayed in the toolbar while the command is executed. Calling `MenuEraseStatus` on Palm OS 3.5 and higher deallocates the command toolbar data structure as well as erasing the command toolbar from the screen.

Because the command toolbar takes up more of the display than the pre-Palm OS 3.5 status message, you may find you need to call `MenuEraseStatus` more frequently in Palm OS 3.5 than in earlier versions.

**See Also** [MenuInit](#)

## MenuGetActiveMenu

**Purpose** Returns a pointer to the currently active menu.

**Declared In** Menu.h

**Prototype** `MenuBarType *MenuGetActiveMenu (void)`

**Parameters** None.

**Result** Returns a pointer to the currently active menu, or `NULL` if there is none.

**Comments** An active menu is not necessarily visible on the screen. A menu might be active but not visible, for example, if a command shortcut has been entered. In general, a form's menu becomes active the first time a [keyDownEvent](#) with a `vchrMenu` or `vchrCommand` is generated, and it remains active until a new form (including a modal form or alert panel) is displayed or until [FrmSetMenu](#) is called to change the form's menu.

Applications that perform custom drawing to a window often check to see if the menu is visible to ensure that they don't draw on top of the menu. See “[Checking Menu Visibility](#)” on page 107 of the *Palm*

## Menus

### Menu Functions

---

*OS Programmer's Companion*, vol. I for instructions on how to verify a menu's visibility.

**See Also** [MenuHandleEvent](#), [MenuSetActiveMenu](#)

## MenuHandleEvent

**Purpose** Handles events in the current menu. This routine handles two types of events, [penDownEvent](#) and [keyDownEvent](#).

**Declared In** Menu.h

**Prototype** Boolean MenuHandleEvent (MenuBarType \*menuP,  
EventType \*event, UInt16 \*error)

**Parameters**

-> menuP	Pointer to a MenuBarType data structure.
-> event	Pointer to an EventType structure.
-> error	Error (or 0 if no error). Currently this function always sets error to zero.

**Result** Returns true if the event is handled; that is, if the event is a [penDownEvent](#) within the menu bar or the menu, or the event is a [keyDownEvent](#) that the menu supports. Returns false on any other event.

**Comments** When a [penDownEvent](#) is received in the menu bar, MenuHandleEvent tracks the pen until it comes up. If the pen comes up within the bounds of the menu bar, the selected title is inverted and the appropriate pull-down menu is drawn. Any previous pull-down menu is erased. If the pen comes up outside of the menu bar and pull-down menu, the menu is erased.  
When a penDownEvent is received in a pull-down menu, MenuHandleEvent tracks the pen until it comes up. If the pen comes up within the bounds of the menu, a [menuEvent](#) containing the resource ID of the selected menu item is added to the event queue. If the pen comes up outside of the bounds of the menu and menu bar, the menu is erased.

If a [keyDownEvent](#) is received with a vchrMenu, the menu is drawn if it is not visible and erased if it is visible.

If a keyDownEvent is received with a vchrCommand, a command shortcut is in progress. Command shortcuts are handled differently depending on which version of Palm OS is running. On versions earlier than 3.5, the next keyDownEvent is checked to see if it is a valid menu shortcut. If so, a [menuEvent](#) is added to the event queue.

If a keyDownEvent is received with a vchrCommand on Palm OS 3.5 and higher, MenuHandleEvent enqueues a [menuCmdBarOpenEvent](#) if the command toolbar is not already open. The menuCmdBarOpenEvent provides a chance for applications to add their own buttons to the command toolbar. The next event might be either a keyDownEvent with a character that completes the shortcut or a penDownEvent on one of the buttons on the toolbar. The keyDownEvent is processed as with the earlier releases—if it is a valid menu shortcut, a [menuEvent](#) is added to the event queue. If the next event is a penDownEvent, the pen is tracked until it comes up. If the pen comes up within the bounds of a button, the appropriate action is taken. See the description of [MenuCmdBarAddButton](#) for more information.

In Palm OS version 3.5 or higher, if either the vchrMenu or the vchrCommand event causes a menu to be activated and initialized for the first time, a [menuOpenEvent](#) containing the reason the menu was initialized (menuButtonCause for a vchrMenu or menuCommandCause for a vchrCommand) is added to the event queue, and then the current event is added after it.

## Menus

### *Menu Functions*

---

## MenuHideItem

**Purpose** Makes the specified menu item hidden.

**Declared In** Menu.h

**Prototype** Boolean MenuHideItem (UInt16 id)

**Parameters** -> id ID of the menu item to hide.

**Result** Returns `true` if the `hidden` attribute of the specified item was successfully enabled, `false` otherwise.

**Comments** You should call this function only in response to a [menuOpenEvent](#). This event is generated when the menu is first made active. In general, a form's menu becomes active the first time a [keyDownEvent](#) with a `vchrMenu` or `vchrCommand` is generated, and it remains active until a new form (including a modal form or alert panel) is displayed or until [FrmSetMenu](#) is called to change the form's menu. Palm OS user interface guidelines discourage adding or hiding menu items at any time other than when the menu is first made active.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [MenuShowItem](#)

## Menulnit

<b>Purpose</b>	Loads a menu resource from a resource file.
<b>Declared In</b>	Menu.h
<b>Prototype</b>	MenuBarType *Menulnit (UInt16 resourceId)
<b>Parameters</b>	-> resourceId ID that identifies the menu resource.
<b>Result</b>	Returns the pointer to a memory block allocated to hold the menu resource (a pointer to a <a href="#">MenuBarType</a> ).
<b>Comments</b>	The menu is not usable until <a href="#">MenuSetActiveMenu</a> is called. Typically, you do not need to call this function directly. A form stores the resource ID of the menu associated with it and initializes that menu as necessary. If you want to change the form's menu, call <a href="#">FrmSetMenu</a> , which handles disposing of the form's current menu, associating the new menu with the form, and initializing when needed.
<b>See Also</b>	<a href="#">MenuSetActiveMenu</a> , <a href="#">MenuDispose</a>

## MenuSetActiveMenu

<b>Purpose</b>	Sets the current menu.
<b>Declared In</b>	Menu.h
<b>Prototype</b>	MenuBarType *MenuSetActiveMenu (MenuBarType *menuP)
<b>Parameters</b>	-> menuP Pointer to the memory block that contains the new menu, or NULL for none.
<b>Result</b>	Returns a pointer to the menu that was active before the new menu was set, or NULL if no menu was active.

## Menus

### Menu Functions

---

**Comments** This function sets the active menu but does not associate it with a form. It's recommended that you call [FrmSetMenu](#) instead of `MenuSetActiveMenu`. `FrmSetMenu` sets the active menu, frees the old menu, and associates the newly active menu with the form, which means the menu will be freed when the form is freed.

**See Also** [MenuGetActiveMenu](#)

## MenuSetActiveMenuRscID

**Purpose** Sets the current menu by resource ID.

**Declared In** `Menu.h`

**Prototype** `void MenuSetActiveMenuRscID (UInt16 resourceId)`

**Parameters** `-> resourceId` Resource ID of the menu to be made active.

**Result** Returns nothing.

**Comments** This function is similar to [MenuSetActiveMenu](#) except that you pass the menu's resource ID instead of a pointer to a menu structure. It is used as an optimization; with `MenuSetActiveMenu`, you must initialize the menu before making it active. Potentially, the application may exit before the menu is needed, making this memory allocation unnecessary. `MenuSetActiveMenuRscID` simply stores the resource ID. The next time a menu is requested, the menu is initialized from this resource.

It's recommended that you call [FrmSetMenu](#) instead of calling this function for the reasons given in `MenuSetActiveMenu`.

**Compatibility** Implemented only if “[2.0 New Feature Set](#)” is present.

## MenuShowItem

**Purpose** Makes the specified menu item visible.

**Declared In** Menu.h

**Prototype** Boolean MenuShowItem (UInt16 id)

**Parameters** -> id ID of the menu item to display.

**Result** Returns `true` if the `hidden` attribute of the specified item was successfully disabled, `false` otherwise.

## **Menus**

### *Menu Functions*

---

- Comments** You should call this function only in response to a [menuOpenEvent](#). This event is generated when the menu is first made active. In general, a form's menu becomes active the first time a [keyDownEvent](#) with a vchrMenu or vchrCommand is generated, and it remains active until a new form (including a modal form or alert panel) is displayed or until [FrmSetMenu](#) is called to change the form's menu. Palm OS user interface guidelines discourage adding or hiding menu items at any time other than when the menu is first made active.
- Compatibility** Implemented only if [3.5 New Feature Set](#) is present.
- See Also** [MenuHideItem](#)

# Private Records

---

This chapter describes the private records API as declared in `PrivateRecords.h`. It discusses the following topics:

- [Private Record Data Structures](#)
- [Private Record Functions](#)

## Private Record Data Structures

### **privateRecordViewEnum**

The `privateRecordViewEnum` enumerated type provides the available choices for displaying private records.

```
typedef enum privateRecordViewEnum {  
    showPrivateRecords = 0x00,  
    maskPrivateRecords,  
    hidePrivateRecords  
} privateRecordViewEnum;
```

#### **Value Descriptions**

`showPrivateRecords` Display private records in the user interface.

`maskPrivateRecords` Show a shaded rectangle in place of a private record.

`hidePrivateRecords` Hide private records and provide no indication in the user interface that they exist.

## Private Records

### *Private Record Functions*

---

## Private Record Functions

### SecSelectViewStatus

**Purpose** Display a form that allows the user to select whether to hide, show, or mask private records.

**Declared In** PrivateRecords.h

**Prototype** privateRecordViewEnum SecSelectViewStatus (void)

**Parameters** None

**Result** Returns a constant that indicates which option the user selected. See [privateRecordViewEnum](#).

**Comments** This function displays a dialog that allows users to change the preference prefShowPrivateRecords, which controls how private records are displayed.

When the user taps the OK button in this dialog, [SecVerifyPW](#) is called to see if the user changed the preference setting and, if so, to prompt the user to enter the appropriate password.

After calling this function, your code should check the return value or the value of prefShowPrivateRecords and mask, display, or hide the private records accordingly. See the description of [TblSetRowMasked](#) for a partial example.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

## SecVerifyPW

<b>Purpose</b>	Display a password dialog, verify the password, and change the private records preference.
<b>Declared In</b>	PrivateRecords.h
<b>Prototype</b>	Boolean SecVerifyPW (privateRecordViewEnum newSecLevel)
<b>Parameters</b>	-> newSecLevel The security level (display, hide, or mask) selected on the private records dialog.
<b>Result</b>	Returns true if the prefShowPrivateRecords preference was successfully changed, false if not.
<b>Comments</b>	This function checks newSecLevel against the current value for the preference. If the two values differ and newSecLevel indicates a decrease in security, a dialog is displayed prompting the user to enter a password. (Hidden is considered the most secure, followed by masked. Showing private records is considered the least secure.) If the password is entered successfully, the preference is changed. This function also displays an alert message if the security level has changed to either hidden or masked.
<b>Compatibility</b>	Implemented only if <a href="#">3.5 New Feature Set</a> is present.

## **Private Records**

### *Private Record Functions*

---

# Progress Manager

---

This chapter provides reference material for the Progress Manager.

- [Progress Manager Functions](#)
- [Application-Defined Functions](#)

The header file `Progress.h` declares the API that this chapter describes. For more information on the progress manager, see the section “[Progress Dialogs](#)” in the *Palm OS Programmer’s Companion*, vol. I.

## Progress Manager Functions

### **PrgHandleEvent**

**Purpose** Handles events related to the active progress dialog.

**Declared In** `Progress.h`

**Prototype** Boolean `PrgHandleEvent (ProgressPtr prgGP,  
EventType *eventP)`

**Parameters** `-> prgGP` Pointer to a progress structure created by [PrgStartDialog](#).

`-> eventP` Pointer to an event. You can pass a NULL event to cause this function to immediately call your [PrgCallbackFunc](#) function and then update the dialog (for example, after you call [PrgUpdateDialog](#)).

**Result** Returns `true` if the system handled the event. If it returns `false`, you should check if the user canceled the dialog by calling [PrgUserCancel](#).

## Progress Manager

### Progress Manager Functions

---

<b>Comments</b>	<p>Use this function instead of <a href="#">SysHandleEvent</a> when you have a progress dialog. PrgHandleEvent internally calls SysHandleEvent as needed.</p> <p>Note that the auto power-off feature of the system is automatically disabled when you use this function, unless the dialog is just displaying an error. This function also prevents appStopEvent events.</p> <p>If an update to the dialog is pending (from a call to <a href="#">PrgUpdateDialog</a>, for example) this function handles that event and causes the dialog to be updated. As part of this process, the textCallback function you specified in your call to <a href="#">PrgStartDialog</a> is called. Your textCallback function should set the textP buffer in the PrgCallbackData structure with the new message to be displayed in the progress dialog. Optionally, you can also set the bitmapId field to include an icon in the update dialog. For more information about the textCallback function, see the section "<a href="#">Application-Defined Functions</a>."</p>
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">PrgStartDialog</a> , <a href="#">PrgStopDialog</a> , <a href="#">PrgUpdateDialog</a> , <a href="#">PrgUserCancel</a>

## PrgStartDialog

<b>Purpose</b>	Displays a progress dialog that can be updated.	
<b>Declared In</b>	Progress.h	
<b>Prototype</b>	ProgressPtr PrgStartDialog (const Char *title, PrgCallbackFunc textCallback, void *userDataP)	
<b>Parameters</b>	-> title	Pointer to a title for the progress dialog. Must be a null-terminated string that is no longer than progressMaxTitle (20).
	-> textCallback	Pointer to a callback function that supplies the text and icons for the current progress state. See <a href="#">PrgCallbackFunc</a> .

-> userDataP A pointer to data that you need to access in the callback function. This address gets passed directly to your function.

**Result** A pointer to a progress structure. This pointer must be passed to other progress manager functions and **must** be released by calling [PrgStopDialog](#). NULL is returned if the system is unable to allocate the progress structure.

**Comments** The dialog created by this function can be updated by another process via the [PrgUpdateDialog](#) function. The dialog can contain a Cancel or OK button. The initial dialog defaults to stage 0 and calls the `textCallback` function to get the initial text and icon data for the progress dialog.

This function saves the screen bits behind the progress dialog, and these are restored when you call `PrgStopDialog`. Because of this, you should minimize changes to the screen while the progress dialog is displayed, otherwise, the restored bits may not match with what is currently being displayed.

**Compatibility** This version of the function is available only if [3.2 New Feature Set](#) is present. On earlier systems, `PrgStartDialog` has the prototype shown for [PrgStartDialogV31](#).

**See Also** [PrgHandleEvent](#), [PrgStopDialog](#), [PrgUpdateDialog](#), [PrgUserCancel](#)

## Progress Manager

### *Progress Manager Functions*

---

## PrgStartDialogV31

**Purpose** Displays a progress dialog that can be updated.

**Declared In** Progress.h

**Prototype** ProgressPtr PrgStartDialogV31 (const Char \*title,  
PrgCallbackFunc textCallback)

**Parameters**

-> title	Pointer to a title for the progress dialog. Must be a null-terminated string that is no longer than progressMaxTitle (20).
-> textCallback	Pointer to a callback function that supplies the text and icons for the current progress state. See <a href="#">PrgCallbackFunc</a> .

**Result** A pointer to a progress structure. This pointer must be passed to  
other progress manager functions and **must** be released by calling  
[PrgStopDialog](#). NULL is returned if the system is unable to  
allocate the progress structure.

**Compatibility** This function corresponds to version of [PrgStartDialog](#)  
available on Palm OS® 3.0 and Palm OS 3.1.

**See Also** [PrgHandleEvent](#), [PrgStopDialog](#), [PrgUpdateDialog](#),  
[PrgUserCancel](#)

## PrgStopDialog

<b>Purpose</b>	Releases memory used by the progress dialog and restores the screen to its initial state.					
<b>Declared In</b>	Progress.h					
<b>Prototype</b>	<pre>void PrgStopDialog (ProgressPtr prgP, Boolean force)</pre>					
<b>Parameters</b>	<table><tr><td>-&gt; prgP</td><td>Pointer to a progress structure created by <a href="#">PrgStartDialog</a>.</td></tr><tr><td>-&gt; force</td><td>true removes the progress dialog immediately, false causes the system to wait until the user confirms an error, if one is displayed.</td></tr></table>		-> prgP	Pointer to a progress structure created by <a href="#">PrgStartDialog</a> .	-> force	true removes the progress dialog immediately, false causes the system to wait until the user confirms an error, if one is displayed.
-> prgP	Pointer to a progress structure created by <a href="#">PrgStartDialog</a> .					
-> force	true removes the progress dialog immediately, false causes the system to wait until the user confirms an error, if one is displayed.					
<b>Result</b>	Returns nothing.					
<b>Comments</b>	If the progress dialog is in a state where it is displaying an error message to the user, this function normally waits for the user to confirm the dialog before it removes the dialog. If you specify true for the force parameter, this causes the system to remove the dialog immediately.					
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.					
<b>See Also</b>	<a href="#">PrgHandleEvent</a> , <a href="#">PrgStartDialog</a> , <a href="#">PrgUpdateDialog</a> , <a href="#">PrgUserCancel</a>					

## Progress Manager

### Progress Manager Functions

---

## PrgUpdateDialog

**Purpose** Updates the status of the current progress dialog.

**Declared In** Progress.h

**Prototype** void PrgUpdateDialog (ProgressPtr prgGP,  
                  UInt16 err, UInt16 stage, const Char \*messageP,  
                  Boolean updateNow)

<b>Parameters</b>	-> prgGP	Pointer to a progress structure created by <a href="#">PrgStartDialog</a> .
	-> err	If non-zero, causes the dialog to go into error mode, to display an error message with only an OK button.
	-> stage	Current stage of progress. The callback function can use this to determine the correct string to display in the updated dialog.
	-> messageP	Extra text that may be useful in displaying the progress for this stage. Used by the callback function, which can append it to the base message that is based on the stage.
	-> updateNow	If true, the dialog is immediately updated. Otherwise, the dialog is updated on the next call to <a href="#">PrgHandleEvent</a> .

**Result** Returns nothing.

**Comments** For more information about how the parameters are used and the callback function, see the section "[Application-Defined Functions](#)."

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [PrgHandleEvent](#), [PrgStartDialog](#), [PrgStopDialog](#), [PrgUserCancel](#)

## PrgUserCancel

<b>Purpose</b>	Macro that returns true if the user cancelled the process via the progress dialog.
<b>Declared In</b>	Progress.h
<b>Prototype</b>	PrgUserCancel (prgP)
<b>Parameters</b>	-> prgP Pointer to a progress structure (ProgressPtr) created by <a href="#">PrgStartDialog</a> .
<b>Result</b>	Returns the value of the cancel field in the progress structure (as a UInt16).
<b>Comments</b>	This is a macro you can use to check if the user cancelled the process. If the user did cancel, you can change the progress dialog text to something like "Cancelling," or "Disconnecting," or whatever is appropriate for your application. Then you should cancel the process, end the communication session, or do whatever processing is necessary.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">PrgHandleEvent</a> , <a href="#">PrgStartDialog</a> , <a href="#">PrgStopDialog</a> , <a href="#">PrgUpdateDialog</a>

# Application-Defined Functions

## PrgCallbackFunc

**Purpose** Supplies the text and icons for the current progress state.

**Declared In** Progress.h

**Prototype** Boolean (\*PrgCallbackFunc)  
(PrgCallbackDataPtr cbP)

**Parameters** <-> cbP A pointer to a PrgCallbackData structure.  
See below.

**Result** Returns true if the progress dialog should be updated using the values you specified in the PrgCallbackData structure. If you specify false, the dialog is still updated, but with default status messages. (Returning false is not recommended.)

**Comments** This is a callback function that you specify when you call [PrgStartDialog](#). The callback function is called by [PrgHandleEvent](#) when it needs current progress information to display in the progress dialog.

The system passes this function one parameter, a pointer to a PrgCallbackData structure. Here are the important fields in that data structure (note that -> indicates you set the field in the textCallback function):

<- UInt16 stage Current stage (passed from  
PrgUpdateDialog).

<-> Char \*textP Buffer to hold the text to display in the updated dialog. You might want to look up a message in a resource file, based on the value in the stage field. Also, you should append the additional text in the message field, to form the full string to display. Be sure to include a null terminator at the end of the string you return, and don't exceed the length in textLen.

```
<- UInt16 textLen
    Maximum length of the text buffer textP. Note
    that this value is set for you by the caller. Be
    careful not to exceed this length in textP.

<- Char *message
    Additional text to display in the dialog (from
    the messageP parameter to
    PrgUpdateDialog). This should be no longer
    than progressMaxMessage (128).

<- Err error
    Current error (passed from the err parameter
    to PrgUpdateDialog).

-> UInt16 bitmapId
    Resource ID of the bitmap to display in the
    progress dialog, if any.

<- UInt16 canceled:1
    true if user has pressed the cancel button.

<- UInt16 showDetails:1
    true if user pressed the down arrow button on
    the Palm™ device for more details. (Because this
    is a non-standard user interface technique, you
    shouldn't use this feature to display details that
    users need under normal conditions. It's more
    for debugging purposes.)

-> UInt16 textChanged:1
    If true, then update text (defaults to true).
    You can set this to false to avoid an update to
    the text.

<- UInt16 timedOut:1
    true if update caused by a timeout.
```

## Progress Manager

### *Application-Defined Functions*

---

<-> UInt32 timeout

Timeout in ticks to force next update. After this number of ticks, an update is automatically triggered (which sets the `timedOut` flag). You can use this feature to do a simple animation effect. Note that you must set the timeout for `EvtGetEvent` to a value that is equal to or less than this value, otherwise you won't get update events as frequently as you expect.

-> UInt16 delay:1

If true, delay for one second after updating the dialog. Use this value when you are displaying the final progress message so that users have a chance to see the message before the dialog closes. This field is available only if [3.2 New Feature Set](#) is present.

<- void \*userDataP

A pointer to any application-defined data that the function needs to access. You specify this pointer as a parameter to [PrgStartDialog](#) if the callback function needs to access some application data but does not have access to application globals. This field is available only if [3.2 New Feature Set](#) is present.

In this function, you should set the value of the `textP` buffer to the string you want to display in the progress dialog when it is updated. You can use the value in the `stage` field to look up a message in a string resource. You also might want to append the text in the `message` field to your base string. Typically, the `message` field would contain more dynamic information that depends on a user selection, such as a phone number, device name, or network identifier, etc.

For example, the `PrgUpdateDialog` function might have been called with a `stage` of 1 and a `messageP` parameter value of a phone number string, "555-1212". Based on the `stage`, you might find the string "Dialing" in a string resource, and append the phone number, to form the final text "Dialing 555-1212" that you place in the `text` buffer `textP`.

Keeping the static strings corresponding to various stages in a resource makes it easier to localize your application. More dynamic information can be passed in via the `messageP` parameter to `PrgUpdateDialog`.

---

**NOTE:** This function is called only if the parameters passed to `PrgUpdateDialog` have changed from the last time it was called. If `PrgUpdateDialog` is called twice with exactly the same parameters, the `textCallback` function is called only once.

---

## **Progress Manager**

*Application-Defined Functions*

---

# Scroll Bars

---

This chapter provides reference material for the scroll bar API.

- [Scroll Bar Data Structures](#)
- [Scroll Bar Resources](#)
- [Scroll Bar Functions](#)

The header file `ScrollBar.h` declares the API that this chapter describes. For more information on scroll bars, see the section “[Scroll Bars](#)” on page 137 in the *Palm OS Programmer’s Companion*, vol. I.

## Scroll Bar Data Structures

### **ScrollBarAttrType**

The `ScrollBarAttrType` bit field defines a scroll bar’s visible characteristics.

```
typedef struct {
    UInt16 usable : 1;
    UInt16 visible : 1;
    UInt16 hilighted : 1;
    UInt16 shown : 1;
    UInt16 activeRegion: 4;
} ScrollBarAttrType;
```

## Scroll Bars

### Scroll Bar Data Structures

---

#### Field Descriptions

usable	If not set, the scroll bar object is not considered part of the current interface of the application, and it doesn't appear on screen.
visible	If set, the scroll bar is allowed to be displayed on the screen. If both visible and shown are true, then the scroll bar is actually displayed on the screen.
hilighted	true if either the up arrow or the down arrow is highlighted.
shown	Set if the scroll bar is visible and if maxValue > minValue. (See <a href="#">ScrollBarType</a> .)
activeRegion	The region of the scroll bar that is receiving the pen down events. Possible values are:  sclUpArrow      The up arrow. sclDownArrow     The down arrow. sclUpPage        The region between the scroll car and the up arrow. sclDownPage      The region between the scroll car and the down arrow. sclCar           The scroll car.

## ScrollBarPtr

The ScrollBarPtr type defines a pointer to a [ScrollBarType](#) structure.

```
typedef ScrollBarType *ScrollBarPtr;
```

You pass the ScrollBarPtr as an argument to all scroll bar functions. You can obtain the ScrollBarPtr using the function [FrmGetObjectPtr](#) in this way:

```
scrollBarPtr = FrmGetObjectPtr(frm,  
    FrmGetObjectIndex(frm, scrollBarID));
```

where `scrollBarID` is the resource ID assigned when you created the scroll bar.

## ScrollBarType

The `ScrollBarType` represents a scroll bar.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `ScrollBarType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {  
    RectangleType           bounds;  
    UInt16                  id;  
    ScrollBarAttrType       attr;  
    Int16                   value;  
    Int16                   minValue;  
    Int16                   maxValue;  
    Int16                   pageSize;  
    Int16                   penPosInCar;  
    Int16                   savePos;  
} ScrollBarType;
```

Your code should treat the `ScrollBarType` structure as opaque. Use the functions described in this chapter to retrieve and set each value. Do not attempt to change structure member values directly.

## Scroll Bars

### Scroll Bar Data Structures

---

#### Field Descriptions

bounds	Position (using absolute coordinates) and size (in pixels) of the scroll bar on the screen. For example, to access the bounds of an object in a form whose ID is kObjectID: { RectangleType rect; FormPtr formP = FrmGetActiveForm();  FrmGetObjectBounds(formP, FrmGetObjectIndex(formP, kObjectID), &rect); }
id	ID value you specified when you created the scroll bar object.
attr	Scroll bar's attributes. See <a href="#">ScrollBarAttrType</a> .
value	Current value of the scroll bar. This value is used to determine where to position the scroll car (the dark region in the scroll bar that indicates the position in the document). Access with <a href="#">SclGetScrollBar</a> .  The number given is typically a number relative to minValue and maxValue. These values have nothing to do with any physical characteristics of the object that the scroll bar is attached to, such as the number of lines in the object.  This value is typically set to 0 initially and then adjusted programmatically with <a href="#">SclSetScrollBar</a> .
minValue	Minimum value. When value equals minValue, the scroll car is positioned at the very top of the scrolling region. This value is typically 0. Access with <a href="#">SclGetScrollBar</a> .

maxValue	Maximum value. When value equals maxValue, the scroll car is positioned at the very bottom of the scrolling region. This value is typically set to 0 initially and then adjusted programmatically with <a href="#"><u>SclSetScrollBar</u></a> . Access with <a href="#"><u>SclGetScrollBar</u></a> .
pageSize	Number of lines to scroll when user scrolls one page. Access with <a href="#"><u>SclGetScrollBar</u></a> .
penPosInChar	Used internally.
savePos	Used internally.

## Scroll Bar Resources

The [Scroll Bar Resource](#) (tSCL) represents a scroll bar.

## Scroll Bar Functions

### **ScIDrawScrollBar**

<b>Purpose</b>	Draw a scroll bar.
<b>Declared In</b>	ScrollBar.h
<b>Prototype</b>	<code>void ScIDrawScrollBar (ScrollBarType *bar)</code>
<b>Parameters</b>	<code>-&gt; bar</code> Pointer to a scroll bar structure (see <a href="#">ScrollBarType</a> ).
<b>Result</b>	Returns nothing.
<b>Comments</b>	This function is called internally by <a href="#">ScISetScrollBar</a> and <a href="#">FrmDrawForm</a> . You rarely need to call it yourself.
<b>Compatibility</b>	Implemented only if <a href="#">2.0 New Feature Set</a> is present.

### **ScIGetScrollBar**

<b>Purpose</b>	Retrieve a scroll bar's current position, its range, and the size of a page.
<b>Declared In</b>	ScrollBar.h
<b>Prototype</b>	<code>void ScIGetScrollBar (const ScrollBarType *bar, Int16 *valueP, Int16 *minP, Int16 *maxP, Int16 *pageSizePolicyP)</code>
<b>Parameters</b>	<code>-&gt; bar</code> Pointer to a scroll bar structure (see <a href="#">ScrollBarType</a> ).

<- valueP	A value representing the scroll car's current position. (The scroll car is the dark region that indicates the position in the document.)
<-minP	A value representing the top of the user interface object.
<-maxP	A value representing the bottom of the user interface object.
<-pageSizePolicyP	Pointer to size of a page (used when page scrolling).

**Result** Returns the scroll bar's current values in valueP, minP, maxP, and pageSizeP.

**Comments** You might use this function immediately before calling [SclSetScrollBar](#) to update the scroll bar. SclGetScrollBar returns the scroll bar's current values, which you can then adjust as necessary and pass to SclSetScrollBar.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

**See Also** [SclSetScrollBar](#)

## SclHandleEvent

**Purpose** Handles events that affect a scroll bar.

**Declared In** ScrollBar.h

**Prototype** Boolean SclHandleEvent (ScrollBarType \*bar,  
const EventType \*event)

**Parameters** -> bar                    Pointer to a scroll bar structure (see [ScrollBarType](#)).  
                        -> event                Pointer to an event ([EventType](#)).

**Result** Returns true if the event was handled.

## Scroll Bars

### Scroll Bar Functions

---

- Comment** When a [penDownEvent](#) occurs, the scroll bar sends an [sclEnterEvent](#) to the event queue. When an [sclEnterEvent](#) occurs, the scroll bar determines what its new value should be based on which region of the scroll bar is receiving the pen down events. It then sends either an [sclRepeatEvent](#) or an [sclExitEvent](#) to the event queue. When the user holds and drags the scroll bar with the pen, the scroll bar sends a [sclRepeatEvent](#). Applications that implement dynamic scrolling should catch this event and move the text each time one arrives. When the user releases the pen from the scroll bar, the scroll bar sends a [sclExitEvent](#). Applications that implement non-dynamic scrolling should catch this event and move the text when [sclExitEvent](#) arrives. Applications that implement dynamic scrolling can ignore this event.

- Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## SclSetScrollBar

**Purpose** Set the scroll bar's current position, its range, and the size of a page. If the scroll bar is visible and its minimum and maximum values are not equal, it's redrawn.

**Declared In** ScrollBar.h

**Prototype** void SclSetScrollBar (ScrollBarType \*bar,  
Int16 value, Int16 min, Int16 max, Int16 pageSize)

**Parameters**

-> bar	Pointer to a scroll bar structure (see <a href="#">ScrollBarType</a> ).
-> value	The position the scroll car should move to. (The scroll car is the dark region that indicates the position in the document.)
-> min	Minimum value.
-> max	Maximum value.

-> pageSize      Number of lines of text that can be displayed on a the screen at one time (used when page scrolling).

**Result**      Returns nothing. May display a fatal error message if the min parameter is greater than the max parameter.

**Comments**      Call this function when the user adds or deletes text in a field or when a table row is added or deleted.

For scrolling fields, your application should catch the [fldChangedEvent](#) and update the scroll bar at that time.

The max parameter is computed as:

$$\text{number of lines of text} - \text{page size} + \text{overlap}$$

where number of lines of text is the total number of lines or rows needed to display the entire object, page size is the number of lines or rows that can be displayed on the screen at one time, and overlap is the number of lines or rows from the bottom of one page to be visible at the top of the next page.

For example, if you have 100 lines of text and 10 lines show on a page, the max value would be 90 or 91, depending on the overlap. So if value is greater than or equal to 90 or 91, the scroll car is at the very bottom of the scrolling region.

You can use the [FldGetScrollValues](#) function to compute the values you pass for value, min, and max. For example:

```
FldGetScrollValues (fld, &scrollPos,  
    &textHeight,  &fieldHeight);  
  
if (textHeight > fieldHeight)  
    maxValue = textHeight - fieldHeight;  
else if (scrollPos)  
    maxValue = scrollPos;  
else  
    maxValue = 0;  
  
SclSetScrollBar (bar, scrollPos, 0, maxValue,  
    fieldHeight-1);
```

## **Scroll Bars**

### *Scroll Bar Functions*

---

In this case, `textHeight` is the number of lines of text and `fieldHeight` is the page size. No lines overlap when you scroll one page. Notice that if the page size is greater than the lines of text, then `max` equals `min`, which means that the scroll bar is not displayed.

For scrolling tables, there is no equivalent to `FldGetScrollValues`. Your application must scroll the table itself and keep track of the scroll values. See the `ListViewUpdateScrollers` function in the Memo example application (`MemoMain.c`) for an example of setting scroll bar values for a table.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

**See Also** [SclGetScrollBar](#)

## **Scroll Bars**

### *Scroll Bar Functions*

---

# System Dialogs

---

This chapter provides reference material for system dialogs declared in the header files FatalAlert.h, Launcher.h, GraffitiReference.h, and GraffitiUI.h.

## System Dialog Functions

### SysAppLauncherDialog

**Purpose** Display the launcher popup, get a choice, ask the system to launch the selected application, clean up, and leave. If there are no applications to launch, nothing happens.

**Declared In** Launcher.h

**Prototype** void SysAppLauncherDialog()

**Parameters** None.

**Result** The system may be asked to launch an application.

**Comments** Typically, this routine is called by the system as necessary. Most applications do not need to call this function themselves.

In Palm OS® version 3.0 and higher the launcher is an application, rather than a popup. This function remains available for compatibility purposes only.

**See Also** [SysAppLaunch](#), the “[Application Launcher](#)” section in the *Palm OS Programmer’s Companion*, vol. I

## **System Dialogs**

### *System Dialog Functions*

---

## **SysFatalAlert**

**Purpose** Display a fatal alert until the user taps a button in the alert.

**Declared In** FatalAlert.h

**Prototype** UInt16 SysFatalAlert (const Char \*msg)

**Parameters** msg Message to display in the dialog.

**Result** The button tapped; first button is zero.

## **SysGraffitiReferenceDialog**

**Purpose** Pop up the Graffiti® Reference Dialog.

**Declared In** GraffitiReference.h

**Prototype** void SysGraffitiReferenceDialog  
(ReferenceType referenceType)

**Parameters** referenceType Which reference to display. See  
GraffitiReference.h for more  
information.

**Result** Nothing returned.

# Tables

---

This chapter describes the table API as declared in the header file `Table.h`. It discusses the following topics:

- [Table Data Structures](#)
- [Table Constants](#)
- [Table Resource](#)
- [Table Functions](#)
- [Application-Defined Functions](#)

For more information on tables, see the section “[Tables](#)” in the *Palm OS Programmer’s Companion*, vol. I.

## Table Data Structures

### TableAttrType

The `TableAttrType` bit field defines the visible characteristics of the table.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `TableAttrType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    UInt16 visible:1;
    UInt16 editable:1;
```

## Tables

### Table Data Structures

---

```
    UInt16 editing:1;
    UInt16 selected:1;
    UInt16 hasScrollBar:1;
    UInt16 reserved:11;
} TableAttrType;
```

Your code should treat the TableAttrType bit field as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change member values directly.

#### Field Descriptions

visible	If set, the table is drawn on the screen. The value of this field is set by <a href="#">TblDrawTable</a> and cleared by <a href="#">TblEraseTable</a> .
editable	If set, the user can modify the table. You specify this when you create the table resource.
editing	If set, the table is in edit mode. The table is in edit mode while the user edits a text item. The value of this field is returned by <a href="#">TblEditing</a> .
selected	If set, the current item (as identified by the <a href="#">TableType</a> fields <code>currentRow</code> and <code>currentColumn</code> ) is selected. Use <a href="#">TblGetSelection</a> to retrieve this value.
hasScrollBar	If set, the table has a scroll bar. Note that this attribute can only be set programmatically. See <a href="#">TblHasScrollBar</a> .

## TableColumnAttrType

The TableColumnAttrType structure defines a column in a table.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the TableColumnAttrType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    Coord           width;
    UInt16          reserved1 : 5;
    UInt16          masked : 1;
    UInt16          editIndicator : 1;
    UInt16          usable : 1;
    UInt16          reserved2 : 8;
    Coord           spacing;
    TableDrawItemFuncPtr drawCallback;
    TableLoadDataFuncPtr loadDataCallback;
    TableSaveDataFuncPtr saveDataCallback;
} TableColumnAttrType;
```

Your code should treat the TableColumnAttrType structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

### Field Descriptions

width	The column's width in pixels. See <a href="#">TblGetColumnWidth</a> and <a href="#">TblSetColumnWidth</a> .
reserved1	Reserved for future use.
masked	If true and the item's row also has a masked attribute of true, the table cell is drawn on the screen but is shaded to obscure the information that it contains. See <a href="#">TblSetColumnMasked</a> .
editIndicator	If true, items in the column should be highlighted if selected while in edit mode. If false, items in the column should not be highlighted. By default, text field items are highlighted in edit mode, but all other types of items are not highlighted. The default can be overridden with <a href="#">TblSetColumnEditIndicator</a> .

## Tables

### Table Data Structures

---

usable	If <code>false</code> , the column is not considered part of the current interface of the application, and it doesn't appear on screen. See <a href="#">TblSetColumnUsable</a> .
reserved2	Reserved for future use.
spacing	The spacing in pixels between this column and the next column. See <a href="#">TblGetColumnSpacing</a> and <a href="#">TblSetColumnSpacing</a> .
drawCallback	Pointer to a function that draws custom items in the column. This function is called during <a href="#">TblDrawTable</a> and <a href="#">TblRedrawTable</a> . See <a href="#">TblSetCustomDrawProcedure</a> .
loadDataCallback	Pointer to a function that loads data into the column. This function is called during <a href="#">TblDrawTable</a> and <a href="#">TblRedrawTable</a> . See <a href="#">TblSetLoadDataProcedure</a> .
saveDataCallback	Pointer to a function that saves the data in the column. Called when the focus moves from one table cell to another and when the table loses focus entirely. See <a href="#">TblSetSaveDataProcedure</a> .

**Compatibility** The `masked` field is defined only if [3.5 New Feature Set](#) is present.

## TableItemPtr

A `TableItemPtr` points to a [TableItemType](#).

```
typedef TableItemType *TableItemPtr;
```

## TableItemType

The `TableItemType` structure defines an item, or cell, within the table.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `TableItemType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    TableItemStyleType itemType;
    FontID             fontID;
    Int16              intValue;
    Char               *ptr;
} TableItemType;
```

Your code should treat the `TableItemType` structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

---

**NOTE:** None of the table items create memory that you need to free. The table manager handles all of the allocating and deallocating of memory for table items. The only memory you are responsible for freeing is the memory handle containing the text that you want displayed in editable text fields. (See [TableLoadDataFuncType](#).)

---

## Field Descriptions

`itemType` The type of the item, such as a control, a text label, and so on. [TblSetItemStyle](#) sets this value. The rest of the fields in this struct are either used or not used depending on the `itemType`. See [Table 20.1](#).

`fontID` ID of the font used to display the item's text. [TblGetItemFont](#) and [TblSetItemFont](#) retrieve and set this value.

## Tables

### Table Data Structures

---

intValue	Integer value of the item. <a href="#">TblGetIntInt</a> and <a href="#">TblSetItemInt</a> retrieve and set this value.
ptr	Pointer to the item's text. <a href="#">TblGetItemPtr</a> and <a href="#">TblSetItemPtr</a> retrieve and set this value. All text items have a maximum of <code>tableMaxTextItemSize</code> .

The following table lists the possible values for the `itemType` field, describes how each type is drawn, describes which other fields are used for each `itemType`, and provides special instructions for setting those fields. Note in particular that the `fontID` field is often not used. Instead, certain items are displayed in a standard font. These are noted in the last column of this table.

**Table 20.1 TableItemType fields**

ItemType	Description	TableItemType Fields Used
checkboxTableItem	A checkbox control.	intValue
customTableItem	Application-defined cell. The height of the item is fixed at 11 pixels.	None. Custom items are drawn using the custom drawing function that you implement. See <a href="#">TableDrawItemFuncType</a> . If you want, you can store data in the <code>intValue</code> and <code>ptr</code> fields.
dateTableItem	Non-editable date in the form <i>month/day</i> , or a dash if the date value is -1. The date is followed by an exclamation point if it has past.	intValue The <code>intValue</code> field should be a value that can be cast to <code>DateType</code> . <code>DateType</code> is currently defined as a 16-bit number: <code>yyyyyyymmmddddd</code> The first 7 bits are the year given as the offset since 1904, the next 4 bits are the month, and the next 5 bits are the day. Dates are always drawn in the current font.

**Table 20.1 TableItemType fields (*continued*)**

<b>ItemType</b>	<b>Description</b>	<b>TableItemType Fields Used</b>
labelTableItem	Non-editable text.	ptr Labels are displayed in the system's default font along with a terminating colon character (':'). Use a customTableItem or tallCustomTableItem if you don't want a colon.
numericTableItem	Non-editable number.	intValue Numbers are displayed in the system's default bold font.
popupTriggerTableItem	A list with a pop-up trigger.	intValue ptr intValue is the index of the list item that should be displayed. ptr is a pointer to the list. Lists are displayed in the system's default font.
tallCustomTableItem	Application-defined cell. The height of the item is equal to the height of the row in which the item is located. This table item type was added in Palm OS 4.0 and is only supported if <a href="#">4.0 New Feature Set</a> is defined.	None. Custom items are drawn using the custom drawing function that you implement. See <a href="#">TableDrawItemFuncType</a> . If you want, you can store data in the intValue and ptr fields.

## Tables

### *Table Data Structures*

---

**Table 20.1 TableItemType fields (*continued*)**

<b>ItemType</b>	<b>Description</b>	<b>TableItemType Fields Used</b>
textTableItem	Editable text field.	fontID ptr For this item type, implement the callback function <a href="#">TableLoadDataFuncType</a> to load text into the table cell and implement the callback <a href="#">TableSaveDataFuncType</a> to save data before the field is freed.
textWithNoteTableItem	Editable text field and a note icon to the right of the text.	fontID ptr For this item type, implement the callback function <a href="#">TableLoadDataFuncType</a> to load text into the table cell and implement the callback <a href="#">TableSaveDataFuncType</a> to save data before the field is freed.

**Table 20.1 TableItemType fields (*continued*)**

<b>ItemType</b>	<b>Description</b>	<b>TableItemType Fields Used</b>
timeTableItem	Not implemented.	
narrowTextTableItem	Editable text with space reserved on the right side of the cell.	fontID ptr intValue intValue is the number of pixels to reserve on the right side of the cell. For this item type, implement the callback function <a href="#">TableDrawItemFuncType</a> to draw in the space reserved on the right side of the cell, the <a href="#">TableLoadDataFuncType</a> callback function to load text into the table cell, and the callback function <a href="#">TableSaveDataFuncType</a> to save data before the field is freed.

## TablePtr

The TablePtr type defines a pointer to a [TableType](#).

```
typedef TableType *TablePtr;
```

You pass the table's pointer as an argument to all table functions.

You can obtain the table's pointer using the function

[FrmGetObjectPtr](#) in this way:

```
tblPtr = FrmGetObjectPtr(frm,  
                         FrmGetObjectIndex(frm, tblID));
```

where `tblID` is the resource ID assigned when you created the table.

## Tables

### Table Data Structures

---

## TableRowAttrType

The TableRowAttrType structure defines a row in a table.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the TableRowAttrType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct {
    UInt16          id;
    Coord           height;
    UInt32          data;
    UInt16          reserved1 : 7;
    UInt16          usable : 1;
    UInt16          reserved2 : 4;
    UInt16          masked : 1;
    UInt16          invalid : 1;
    UInt16          staticHeight : 1;
    UInt16          selectable : 1;
    UInt16          reserved3;
} TableRowAttrType;
```

Your code should treat the TableRowAttrType structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

### Field Descriptions

id	The ID of this row. See <a href="#">TblFindRowID</a> , <a href="#">TblGetRowID</a> , and <a href="#">TblSetRowID</a> .
height	Height of the row in pixels. The functions <a href="#">TblSetRowHeight</a> and <a href="#">TblGetRowHeight</a> set and retrieve this value.

data	Any application-specific value you want to store in this row. For example, the Datebook and ToDo applications use this field to store the unique ID of the database record that is displayed in this table row. See <a href="#">TblFindRowData</a> , <a href="#">TblGetRowData</a> , and <a href="#">TblSetRowData</a> .
reserved1	Reserved for future use.
usable	If false, the row is not considered part of the current interface of the application, and it doesn't appear on screen. Table rows have usable set to false when they are scrolled off the screen. See <a href="#">TblRowUsable</a> and <a href="#">TblSetRowUsable</a> . The function <a href="#">TblGetLastUsableRow</a> returns the row that appears at the bottom of the screen.
masked	If true and the item's column also has a masked attribute of true, the table cell is drawn on the screen but is shaded to obscure the information that it contains. See <a href="#">TblSetRowMasked</a> and <a href="#">TblRowMasked</a> .
reserved2	Reserved for future use.
invalid	If true, the row needs to be redrawn. See <a href="#">TblRowInvalid</a> , <a href="#">TblMarkRowInvalid</a> , and <a href="#">TblMarkTableInvalid</a> .
staticHeight	true if the row height cannot be changed, false otherwise. If false, text fields in this table row will dynamically resize to multiple lines as necessary. See <a href="#">TblSetRowStaticHeight</a> .
selectable	If true, the user can select individual cells in this row. See <a href="#">TblSetRowSelectable</a> and <a href="#">TblRowSelectable</a> .
reserved3	Reserved for future use.

**Compatibility** The masked field is defined only if [3.5 New Feature Set](#) is present.

## Tables

### Table Data Structures

---

## TableType

The TableType structure represents a table.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the TableType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct TableType {  
    UInt16 id;  
    RectangleType bounds;  
    TableAttrType attr;  
    Int16 numColumns;  
    Int16 numRows;  
    Int16 currentRow;  
    Int16 currentColumn;  
    Int16 topRow;  
    TableColumnAttrType *columnAttrs;  
    TableRowAttrType *rowAttrs;  
    TableItemPtr items;  
    FieldType currentField;  
} TableType;
```

Your code should treat the TableType structure as opaque. Use the functions specified in the descriptions below to retrieve and set each value. Do not attempt to change structure member values directly.

### Field Descriptions

id	ID value you specified when you created the table resource. This ID is included as part of the event data of <a href="#">tblEnterEvent</a> .
bounds	Position and size of the table object. The functions <a href="#">TblGetBounds</a> , <a href="#">FrmGetObjectBounds</a> , <a href="#">TblSetBounds</a> , and <a href="#">FrmSetObjectBounds</a> retrieve and set this value.

attr	The table's attributes. See <a href="#">TableAttrType</a> .
numColumns	Number of columns displayed by the table object. You specify the number of columns when you create the table resource, and this value cannot be changed. Access with <a href="#">TblGetNumberOfColumns</a> .
numRows	Maximum number of visible rows in the table object. You specify this value when you create the table resource, and it does not change; however, the total number of rows in a table can change if you insert new rows in a table, and even the number of currently visible rows can change if a text field within a table cell is dynamically resized. The function <a href="#">TblGetNumberOfRows</a> returns the value of this field.
currentRow	Row index of the currently selected table cell. Rows are numbered from top to bottom starting with 0. <a href="#">TblGetSelection</a> and <a href="#">TblSetSelection</a> retrieve and set the values of currentRow.
currentColumn	Column index of the currently selected table cell. Columns are numbered from left to right starting with 0. If the <a href="#">TableAttrType</a> selected is true, then this table cell is highlighted. If selected is false, the table still considers this the "current" item, but it is not highlighted. <a href="#">TblGetSelection</a> and <a href="#">TblSetSelection</a> retrieve and set the values of currentColumn.
topRow	First visible row of the table object. Access with <a href="#">TblGetTopRow</a> .
columnAttrs	An array of each table column's attributes. See <a href="#">TableColumnAttrType</a> .

## Tables

### Table Constants

---

rowAttrs	An array of each row's attributes, such as its ID, height, and whether or not it is usable, selectable, or invalid. See <a href="#">TableRowAttrType</a> .
items	An array of each item's (table cell's) attributes, such as the item type, font ID, an integer value, and a character pointer. See <a href="#">TableItemType</a> .
currentField	Field object the user is currently editing. The function <a href="#">TblGetCurrentField</a> retrieves the value of this item.

## Table Constants

Constant	Value	Description
tableDefaultColumnSpacing	1	Never used.
tableNoteIndicatorHeight	11	The height in pixels of the note indicator for tables items of type <code>textWithNoteTableItem</code> .
tableNoteIndicatorWidth	7	The width in pixels of the note indicator for tables items of type <code>textWithNoteTableItem</code> .
tableMaxTextItemSize	255	The maximum length of an editable text field within a table cell.
tblUnusableRow	0xffff	Value returned by <a href="#">TblGetLastUsableRow</a> if none of the table's rows are usable. This value is only available in version 3.5 and higher.

## Table Resource

The [Table Resource](#) (tTBL) represents a table on screen.

# Table Functions

## TblDrawTable

**Purpose** Draw a table.

**Declared In** Table.h

**Prototype** void TblDrawTable (TableType \*tableP)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)

**Result** Returns nothing.

**Comments** This function is called as part of [FrmDrawForm](#) when the form contains a table object.

This function draws the entire table, marking all rows valid before drawing. See the [TableItemType](#) struct description for more information about how each type of table cell is drawn.

When drawing cells with editable text fields (textTableItem, textWithNoteTableItem, or narrowTextTableItem), this function uses the [TableLoadDataFuncType](#) callback function to load the text into the table cells. The text field does not retain the text handle that your TableLoadDataFunc returns, meaning that you are responsible for freeing the memory that you load into the table.

When drawing narrowTextTableItem cells, customTableItem cells or tallCustomTableItem cells, this function uses the [TableDrawItemFuncType](#) callback function to draw the extra pixels after the text or to draw the entire cell.

On color systems, tables are always drawn using the same color as is used for the field background color.

When the user has set the security setting to mask private records, table cells that contain private database records are drawn as shaded cells to obscure the information they contain. You must explicitly tell the table which cells are masked by making the

## Tables

### Table Functions

---

appropriate calls to [TblSetRowMasked](#) and [TblSetColumnMasked](#).

<b>Compatibility</b>	Color support and masked private records are only supported in Palm OS® version 3.5 and higher.  In versions earlier than 3.5, this function did not erase table cells before it drew them. In earlier releases, consider calling <a href="#">TblEraseTable</a> before calling this function, particularly if the entire table has changed, as the visual effect of drawing over a white background may be more pleasing.
----------------------	---

<b>See Also</b>	<a href="#">TblEraseTable</a> , <a href="#">TblRedrawTable</a> , <a href="#">TblSetCustomDrawProcedure</a>
-----------------	---

## TblEditing

<b>Purpose</b>	Check whether a table is in edit mode.
<b>Declared In</b>	Table.h
<b>Prototype</b>	Boolean TblEditing (const TableType *tableP)
<b>Parameters</b>	-> tableP              Pointer to a table object. (See <a href="#">TableType</a> .)
<b>Result</b>	Returns <code>true</code> if the table is in edit mode, <code>false</code> otherwise.
<b>Comments</b>	The table is in edit mode while the user edits a text item. More specifically, the table is in edit mode when a <a href="#">tblEnterEvent</a> is received on an editable table cell ( <code>textTableItem</code> , <code>textWithNoteTableItem</code> , or <code>narrowTextTableItem</code> ), or when <a href="#">TblGrabFocus</a> is called.  The table is taken out of edit mode when a the user places the pen on a note in a <code>textWithNoteTableItem</code> or when the table releases the focus ( <a href="#">TblReleaseFocus</a> ).

## TblEraseTable

<b>Purpose</b>	Erase a table object.
<b>Declared In</b>	Table.h
<b>Prototype</b>	void TblEraseTable (TableType *tableP)
<b>Parameters</b>	-> tableP      Pointer to a table object. (See <a href="#">TableType</a> .)
<b>Result</b>	Returns nothing.
<b>Comments</b>	This function sets the table's visible and selected attributes to false. It does not invalidate table rows.
<b>See Also</b>	<a href="#">TblDrawTable</a> , <a href="#">TblSetCustomDrawProcedure</a> , <a href="#">TblRedrawTable</a>

## TblFindRowData

<b>Purpose</b>	Return the number of the row that contains the specified data value.
<b>Declared In</b>	Table.h
<b>Prototype</b>	Boolean TblFindRowData (const TableType *tableP, UInt32 data, Int16 *rowP)
<b>Parameters</b>	-> tableP      Pointer to a table object. (See <a href="#">TableType</a> .) -> data           Row data to find. <- rowP          Pointer to the row number (return value).
<b>Result</b>	Returns true if a match was found, false otherwise.

## Tables

### Table Functions

---

**Comments** This function searches for a row whose attributes contain the specified data. The data is any application-specific data that you have set with [TblSetRowData](#).

**See Also** [TblGetRowData](#), [TblFindRowID](#), [TableRowAttrType](#)

## TblFindRowID

**Purpose** Return the number of the row with the specified ID.

**Declared In** Table.h

**Prototype** Boolean TblFindRowID (const TableType \*tableP,  
                  UInt16 id, Int16 \*rowP)

**Parameters** -> tableP         Pointer to a table object. (See [TableType](#).)  
                -> id                 Row ID to find.  
                -<- rowP             Pointer to the row number (return value).

**Result** Returns true if a match was found, false otherwise.

**See Also** [TblSetRowID](#), [TblFindRowData](#), [TableRowAttrType](#)

## TblGetBounds

**Purpose** Return the bounds of a table.

**Declared In** Table.h

**Prototype** void TblGetBounds (const TableType \*tableP,  
                  RectangleType \*rP)

**Parameters** -> tableP         Pointer to a table object. (See [TableType](#).)

<- rP                    A RectangleType structure in which the bounds are returned.

**Result**    Returns nothing. The rP parameter contains the bounds.

**See Also**    [TblGetItemBounds](#)

## TblGetColumnSpacing

**Purpose**    Return the spacing after the specified column.

**Declared In**    Table.h

**Prototype**    `Coord TblGetColumnSpacing  
(const TableType *tableP, Int16 column)`

**Parameters**    -> tableP            Pointer to a table object. (See [TableType](#).)  
                    -> column            Column number (zero-based).

**Result**    Returns the spacing after column (in pixels).

This function may display a fatal error message if the column parameter is invalid.

**See Also**    [TblGetColumnWidth](#), [TblSetColumnSpacing](#),  
[TblSetColumnUsable](#)

## TblGetColumnWidth

**Purpose**    Return the width of the specified column.

**Declared In**    Table.h

**Prototype**    `Coord TblGetColumnWidth (const TableType *tableP,  
                    Int16 column)`

**Parameters**    -> tableP            Pointer to a table object. (See [TableType](#).)

## Tables

### Table Functions

---

-> column	Column number (zero-based).
<b>Result</b>	Returns the width of a column (in pixels). This function may display a fatal error message if the <code>column</code> parameter is invalid.
<b>See Also</b>	<a href="#">TblGetColumnSpacing</a> , <a href="#">TblSetColumnWidth</a> , <a href="#">TblSetColumnUsable</a>

## TblGetCurrentField

<b>Purpose</b>	Return a pointer to the <a href="#">FieldType</a> in which the user is currently editing a text item.
<b>Declared In</b>	<code>Table.h</code>
<b>Prototype</b>	<code>FieldPtr TblGetCurrentField (const TableType *tableP)</code>
<b>Parameters</b>	<code>-&gt; tableP</code> Pointer to a table object. (See <a href="#">TableType</a> .)
<b>Result</b>	Returns a pointer to the currently selected field, or NULL if the table is not in edit mode.
<b>See Also</b>	<a href="#">TblGetSelection</a>

## TblGetItemBounds

<b>Purpose</b>	Return the bounds of an item in a table.
<b>Declared In</b>	<code>Table.h</code>
<b>Prototype</b>	<code>void TblGetItemBounds (const TableType *tableP, Int16 row, Int16 column, RectangleType *rP)</code>
<b>Parameters</b>	<code>-&gt; tableP</code> Pointer to a table object. (See <a href="#">TableType</a> .)
	<code>-&gt; row</code> Row number of the item (zero-based).
	<code>-&gt; column</code> Column number of the item (zero-based).

<- rP                    Pointer to a structure that holds the bounds of the item.

**Result**    Returns nothing. Stores the bounds in rP. This function may raise a fatal exception if the row or column parameter specifies a row or column that does not appear on screen.

## TblGetItemFont

**Purpose**    Return the font used to display a table item.

**Declared In**    Table.h

**Prototype**    `FontID TblGetItemFont (const TableType *tableP,  
                  Int16 row, Int16 column)`

**Parameters**

-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)
-> row	Row number of the item (zero-based).
-> column	Column number of the item (zero-based).

**Result**    Returns the ID of the font used for the table item at the row and column indicated. This function may display a fatal error message if the row or column parameter specifies a row or column that is not on the screen.

**Comments**    This function returns the value stored in the fontID field for this table item. Only certain types of table items use the font specified by the fontID field when they are displayed. The [TableItemType](#) description specifies what font is used to display each type of table item.

**Compatibility**    Implemented only if [3.0 New Feature Set](#) is present.

**See Also**    [TblSetFont](#)

## Tables

### Table Functions

---

## TblGetItemInt

<b>Purpose</b>	Return the integer value stored in a table item.						
<b>Declared In</b>	Table.h						
<b>Prototype</b>	<code>Int16 TblGetItemInt (const TableType *tableP, Int16 row, Int16 column)</code>						
<b>Parameters</b>	<table><tr><td><code>-&gt; tableP</code></td><td>Pointer to a table object. (See <a href="#">TableType</a>.)</td></tr><tr><td><code>-&gt; row</code></td><td>Row number of the item (zero-based).</td></tr><tr><td><code>-&gt; column</code></td><td>Column number of the item (zero-based).</td></tr></table>	<code>-&gt; tableP</code>	Pointer to a table object. (See <a href="#">TableType</a> .)	<code>-&gt; row</code>	Row number of the item (zero-based).	<code>-&gt; column</code>	Column number of the item (zero-based).
<code>-&gt; tableP</code>	Pointer to a table object. (See <a href="#">TableType</a> .)						
<code>-&gt; row</code>	Row number of the item (zero-based).						
<code>-&gt; column</code>	Column number of the item (zero-based).						
<b>Result</b>	Returns the integer value. This function may display a fatal message if the <code>row</code> or <code>column</code> does not appear on the screen.						
<b>Comments</b>	This function returns the value stored in the <code>intValue</code> field for this table item. Certain types of table items display the value stored in <code>intValue</code> , and other types display the value pointed to by the <code>ptr</code> field. See the <a href="#">TableItemType</a> description for details. If the <code>intValue</code> was never set for this table item, this function returns 0.						
<b>See Also</b>	<a href="#">TblSetItemInt</a> , <a href="#">TblGetItemPtr</a>						

## TblGetItemPtr

<b>Purpose</b>	Return the pointer value stored in a table item				
<b>Declared In</b>	Table.h				
<b>Prototype</b>	<code>void *TblGetItemPtr (const TableType *tableP, Int16 row, Int16 column)</code>				
<b>Parameters</b>	<table><tr><td><code>-&gt; tableP</code></td><td>Pointer to a table object. (See <a href="#">TableType</a>.)</td></tr><tr><td><code>-&gt; row</code></td><td>Row number of the item (zero-based).</td></tr></table>	<code>-&gt; tableP</code>	Pointer to a table object. (See <a href="#">TableType</a> .)	<code>-&gt; row</code>	Row number of the item (zero-based).
<code>-&gt; tableP</code>	Pointer to a table object. (See <a href="#">TableType</a> .)				
<code>-&gt; row</code>	Row number of the item (zero-based).				

	-> column	Column number of the item (zero-based).
<b>Result</b>	Returns the item's pointer value or NULL if the item does not have a pointer value. This function may display a fatal message if the row or column parameter is invalid.	
<b>Comments</b>	This function returns the value stored in the ptr field for this table item. Certain types of table items display the value pointed to by the ptr, and other types display the value stored in the intValue field. See the <a href="#">TableItemType</a> description for details. An application may have set the value of the ptr field anyway, even for items that use the intValue. This function always returns that value.	
<b>Compatibility</b>	Implemented only if <a href="#">3.5 New Feature Set</a> is present. In earlier versions, you can implement this function using the following code:	<pre>return tableP-&gt;items [row * tableP-&gt;numColumns + column] .ptr;</pre>
<b>See Also</b>	<a href="#">TblSetItemPtr</a>	

## TblGetLastUsableRow

<b>Purpose</b>	Return the last row in a table that is usable (visible).	
<b>Declared In</b>	<code>Table.h</code>	
<b>Prototype</b>	<code>Int16 TblGetLastUsableRow (const TableType *tableP)</code>	
<b>Parameters</b>	-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)
<b>Result</b>	Returns the row index (zero-based) or tblUnusableRow if there are no usable rows.	
<b>See Also</b>	<a href="#">TblGetRowData</a> , <a href="#">TblGetRowID</a>	

## Tables

### Table Functions

---

## TblGetNumberOfColumns

<b>Purpose</b>	Return the number of columns in a table.
<b>Declared In</b>	Table.h
<b>Prototype</b>	<code>Int16 TblGetNumberOfColumns (const TableType *tableP)</code>
<b>Parameters</b>	<code>-&gt; tableP</code> Pointer to a <a href="#">TableType</a> .
<b>Result</b>	This function returns the number of columns in a table.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call <code>TblGlueGetNumberOfColumns</code> . For more information, see <a href="#">Chapter 75, “PalmOSGlue Library.”</a>
<b>See Also</b>	<a href="#">TblGetTopRow</a> , <a href="#">TblSetSelection</a>

## TblGetNumberOfRows

<b>Purpose</b>	Return the number of rows in a table.
<b>Declared In</b>	Table.h
<b>Prototype</b>	<code>Int16 TblGetNumberOfRows (const TableType *tableP)</code>
<b>Parameters</b>	<code>-&gt; tableP</code> Pointer to a table object. (See <a href="#">TableType</a> .)
<b>Result</b>	Returns the maximum number of visible rows in the specified table.
<b>Comments</b>	Note that even though you can add and remove rows to and from a table, the value returned by this function does not change. The value returned by this function indicates the maximum number of

rows that can be displayed on the screen at one time. It is set when you create the table resource.

## TblGetRowData

**Purpose** Return the data value of the specified row.

**Declared In** Table.h

**Prototype** UInt32 TblGetRowData (const TableType \*tableP,  
Int16 row)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)  
-> row Number of the row (zero-based).

**Result** Returns the application-specific data stored for this row, if any.  
Returns 0 if there is no application-specific data value.

This function may display a fatal error message if the `row` parameter is invalid.

**See Also** [TblFindRowData](#), [TblSetRowData](#), [TableRowAttrType](#)

## TblGetRowHeight

**Purpose** Return the height of the specified row.

**Declared In** Table.h

**Prototype** Coord TblGetRowHeight (const TableType \*tableP,  
Int16 row)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)

## Tables

### Table Functions

---

-> row                    Number of the row (zero-based).

**Result**    Returns the height in pixels. This function may display a fatal error message if the `row` parameter is invalid.

**See Also**    [TblGetItemBounds](#), [TblSetRowHeight](#)

## TblGetRowID

**Purpose**    Return the ID value of the specified row.

**Declared In**    Table.h

**Prototype**    `UInt16 TblGetRowID (const TableType *tableP,  
                  Int16 row)`

**Parameters**    -> `tableP`                    Pointer to a table object. (See [TableType](#).)  
                  -> `row`                        Number of the row (zero-based).

**Result**    Returns the ID value of the row in the table.

This function may display a fatal error message if the `row` parameter is invalid.

**See Also**    [TblGetRowData](#), [TblSetRowID](#), [TblFindRowID](#),  
[TableRowAttrType](#)

## TblGetSelection

**Purpose**    Return the row and column of the currently selected table item.

**Declared In**    Table.h

**Prototype**    `Boolean TblGetSelection (const TableType *tableP,  
                  Int16 *rowP, Int16 *columnP)`

**Parameters**    -> `tableP`                    Pointer to a table object. (See [TableType](#).)

<- rowP, columnP

The row and column indexes (zero-based) of the currently selected item.

**Result** Returns `true` if the item is highlighted, `false` if not.

**See Also** [TblSetRowSelectable](#)

## TblGetTopRow

**Purpose** Return the top row visible row of a table.

**Declared In** Table.h

**Prototype** Int16 TblGetTopRow(const TableType \*tableP)

**Parameters** -> tableP Pointer to a [TableType](#).

**Result** This function returns the top visible row in a table.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TblGlueGetToRow`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [TblGetNumberOfColumns](#), [TblSetSelection](#)

## TblGrabFocus

**Purpose** Put a table into edit mode.

**Declared In** Table.h

**Prototype** void TblGrabFocus (TableType \*tableP, Int16 row, Int16 column)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)

## Tables

### Table Functions

---

-> row	Current row to be edited (zero-based).
-> column	Current column to be edited (zero-based).

**Result** Returns nothing. This function may display a fatal error message if the table already has the focus or if the `row` or `column` parameter is invalid.

**Comments** This function puts the table into edit mode and sets the current item to the one at the row and column passed in. An editable field must exist in the coordinates passed to this function.

You must call [FrmSetFocus](#) before calling this function. `FrmSetFocus` releases the focus from the object that previously had it and sets the form's internal structures. After calling this function, you must call [TblGrabFocus](#) to display the insertion point in the field. (You can use [TblGetCurrentField](#) to obtain a pointer to the field.)

For example, the following function from the Address Book application sets the focus in an editable field within a table:

```
static void EditViewRestoreEditState () {
    Int16      row;
    FormPtr    frm;
    TablePtr   table;
    FieldPtr   fld;

    if (CurrentFieldIndex == noFieldIndex)
        return;

    // Find the row that the current field is in.
    table = GetObjectPtr (EditTable);
    if ( ! TblFindRowID (table,
        CurrentFieldIndex, &row) )
        return;

    frm = FrmGetActiveForm ();
    FrmSetFocus (frm, FrmGetObjectIndex (frm,
        EditTable));
    TblGrabFocus (table, row, editDataColumn);
```

```
// Restore the insertion point position.  
fld = TblGetCurrentField (table);  
FldSetInsPtPosition (fld, EditFieldPosition);  
FldGrabFocus (fld);  
}
```

**See Also** [TblReleaseFocus](#)

## TblHandleEvent

**Purpose** Handle an event for the table.

**Declared In** Table.h

**Prototype** Boolean TblHandleEvent (TableType \*tableP,  
EventType \*event)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)  
-> event The event to be handled.

**Result** Returns `true` if the event was handled, `false` if it was not.

**Comments** Returns `false` if the table is not an editable table.

If the table is editable, this function passes along any [keyDownEvent](#), [fldEnterEvent](#), or [menuCmdBarOpenEvent](#) to the currently selected field.

When a [fldHeightChangedEvent](#) occurs, this function changes the height of the specified field as indicated by the event. If the field being resized is going to scroll off the bottom of the screen, then instead the table scrolls the rows above it up off the top. Otherwise, the table is scrolled downward and rows below the current row are scrolled off the bottom as necessary.

Note that the `fldHeightChangedEvent` is only handled for dynamically sized fields. See the descriptions of [FieldAttrType](#) and [FldMakeFullyVisible](#) for more information.

When a [penDownEvent](#) occurs, the table checks to see if the focus is being changed. If it is and the user was previously editing a text

## Tables

### Table Functions

---

field within the table, it saves the data in the table cell using the [TableSaveDataFuncType](#) callback function, then it enqueues a [tblEnterEvent](#) with the new row and column that are selected.

When a [tblEnterEvent](#) occurs, this function tracks the pen until it is lifted. If the pen is lifted within the bounds of the same item it went down in, a [tblSelectEvent](#) is added to the event queue; if not, a [tblExitEvent](#) is added to the event queue.

## TblHasScrollBar

**Purpose** Set the hasScrollBar attribute in the table. (See [TableAttrType](#).)

**Declared In** Table.h

**Prototype** void TblHasScrollBar (TableType \*tableP,  
Boolean hasScrollBar)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)  
-> hasScrollBar true to set the attribute, false to unset it.

**Result** Returns nothing.

**Comments** Your application must scroll the table itself and keep track of the scroll values. See the [ListViewUpdateScrollers](#) function in the Memo example application (MemoMain.c) for an example of setting scroll bar values for a table.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## TblInsertRow

**Purpose** Insert a row into the table before the specified row.

**Declared In** Table.h

**Prototype** void TblInsertRow (TableType \*tableP, Int16 row)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)  
-> row Row to insert (zero-based).

**Result** Returns nothing.

**Comments** The number of rows in a table is the maximum number of rows displayed on the screen. Unlike a multi-line text field, there is no notion of a table that is bigger than the available screen. For this reason, this function does not increase the number of table rows. Instead of keeping track of a total number of rows in the table and a number of rows displayed on the screen, tables mark any row that isn't currently displayed with a usable value of `false`. (See [TableRowAttrType](#).) The newly inserted row is marked as invalid, unusable, and not masked. If you want to display the newly inserted row, call [TblSetRowUsable](#) after making sure that the row displays a value and then call [TblRedrawTable](#) when you are ready to draw the table.

**See Also** [TblRemoveRow](#), [TblSetRowUsable](#), [TblSetRowSelectable](#)

## Tables

### Table Functions

---

## TblMarkRowInvalid

**Purpose** Mark the row invalid.

**Declared In** Table.h

**Prototype** void TblMarkRowInvalid (TableType \*tableP,  
Int16 row)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)  
-> row Row number (zero-based).

**Result** Returns nothing.

**Comments** After calling this function, call [TblRedrawTable](#) to redraw all rows marked invalid.

This function may display a fatal error message if the `row` parameter is invalid.

**See Also** [TblRemoveRow](#), [TblSetRowUsable](#), [TblSetRowSelectable](#), [TblMarkTableInvalid](#), [TblRowInvalid](#), [TableRowAttrType](#)

## TblMarkTableInvalid

**Purpose** Mark all the rows in a table invalid.

**Declared In** Table.h

**Prototype** void TblMarkTableInvalid (TableType \*tableP)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)

**Result** Returns nothing.

**Comments** After calling this function, you must call [TblRedrawTable](#) to redraw all rows.

**See Also** [TblEraseTable](#), [TblRedrawTable](#), [TableRowAttrType](#)

## TblRedrawTable

**Purpose** Redraw the rows of the table that are marked invalid.

**Declared In** Table.h

**Prototype** void TblRedrawTable (TableType \*tableP)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)

**Result** Returns nothing.

**Comments** This function draws the invalid rows in the table. See the [TableItemType](#) struct description for more information about how each type of table cell is drawn.

When drawing cells with editable text fields (textTableItem, textWithNoteTableItem, or narrowTextTableItem), this function uses the [TableLoadDataFuncType](#) callback function to load the text into the table cells. The text field does not retain the text handle that your TableLoadDataFunc returns, meaning that you are responsible for freeing the memory that you load into the table.

When drawing narrowTextTableItem cells, customTableItem cells, or tallCustomTableItem cells, this function uses the [TableDrawItemFuncType](#) callback function to draw the extra pixels after the text or to draw the entire cell.

On color systems, tables are always drawn using the same color as is used for the field background color.

When the user has set the security setting to mask private records, table cells that contain private database records are drawn as shaded cells to obscure the information they contain. You must explicitly tell the table which cells are masked by making the

## Tables

### Table Functions

---

appropriate calls to [TblSetRowMasked](#) and [TblSetColumnMasked](#).

**Compatibility** Color support and masked private records are only supported in Palm OS version 3.5 and higher.

**See Also** [TblMarkTableInvalid](#), [TblMarkRowInvalid](#), [TblDrawTable](#)

## TblReleaseFocus

**Purpose** Release the focus.

**Declared In** Table.h

**Prototype** void TblReleaseFocus (TableType \*tableP)

**Parameters** -> tableP Pointer to a table object.

**Result** Returns nothing.

**Comments** You typically do not call this function yourself. Instead, call [FrmSetFocus](#) with an object index of noFocus to notify the form that the table has lost focus. The form code calls TblReleaseFocus for you.

If the current item is a text item, the [TableSaveDataFuncType](#) callback function is called to save the text in the currently selected field, the memory allocated for editing is released, and the insertion point is turned off.

Also note that you might have to call [FldReleaseFocus](#) if the focus is in an editable text field and that field uses a custom drawing function ([TableDrawItemFuncType](#)).

**See Also** [TblGrabFocus](#)

## TblRemoveRow

**Purpose** Remove the specified row from the table.

**Declared In** Table.h

**Prototype** void TblRemoveRow (TableType \*tableP, Int16 row)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)  
-> row Row to remove (zero-based).

**Result** Returns nothing. This function may raise a fatal error message if an invalid row is specified.

**Comments** The number of rows in the table is not decreased; instead, this row is moved from its current spot to the end of the table and is marked unusable so that it won't be displayed when the table is redrawn. This function does not visually update the display. To update the display, call [TblRedrawTable](#).

**See Also** [TblInsertRow](#), [TblSetRowUsable](#), [TblSetRowSelectable](#), [TblMarkRowInvalid](#)

## TblRowInvalid

**Purpose** Return whether a row is invalid.

**Declared In** Table.h

**Prototype** Boolean TblRowInvalid (const TableType \*tableP, Int16 row)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#)).  
-> row Row number (zero-based).

**Result** Returns `true` if the row is invalid, `false` if it's valid. This function may raise a fatal error message if the `row` parameter is invalid.

## Tables

### Table Functions

---

**Comments** Invalid rows need to be redrawn. Use [TblRedrawTable](#) to do so.

**See Also** [TblMarkRowInvalid](#), [TblMarkTableInvalid](#)

## TblRowMasked

**Purpose** Return whether a row is masked.

**Declared In** Table.h

**Prototype** Boolean TblRowMasked (const TableType \*tableP,  
Int16 row)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)  
-> row Row number (zero-based).

**Result** Returns true if the row is masked, false otherwise.

**Comments** Your code should set a row to masked if it contains a private database record and the user has set the display preference for private records to masked. Masked cells are displayed as shaded.

Note that a table cell is not masked unless both its row and column are masked. This allows non-private information in the row item to remain visible. For example, the Datebook application shows the time for a private appointment, but it does not show the description.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [TblSetRowMasked](#), [TblSetColumnMasked](#),  
[TableRowAttrType](#), [SecSelectViewStatus](#)

## TblRowSelectable

**Purpose** Return whether the specified row is selectable.

**Declared In** Table.h

**Prototype** Boolean TblRowSelectable  
(const TableType \*tableP, Int16 row)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)  
-> row Row number (zero-based).

**Result** Returns `true` if the row is selectable, `false` if it's not.

**Comments** Rows that are not selectable don't highlight when touched.

**See Also** [TableRowAttrType](#)

## TblRowUsable

**Purpose** Determine whether the specified row is usable.

**Declared In** Table.h

**Prototype** Boolean TblRowUsable (const TableType \*tableP,  
Int16 row)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)  
-> row Row number (zero-based).

**Result** Returns `true` if the row is usable, `false` if it's not.

This function may display a fatal error message if the `row` parameter is invalid.

## Tables

### Table Functions

---

**Comments** Rows that are not usable do not display.

**See Also** [TblRowSelectable](#), [TblGetLastUsableRow](#),  
[TblSetRowUsable](#)

## TblSelectItem

**Purpose** Select (highlight) the specified item. If there is already a selected item, it is unhighlighted.

**Declared In** Table.h

**Prototype** void TblSelectItem (TableType \*tableP, Int16 row,  
Int16 column)

**Parameters**

-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)
-> row	Row of the item to select (zero-based).
-> column	Column of the item to select (zero-based).

**Result** Returns nothing.

This function may display a fatal error message if the column or row parameter point to an item that is not on the screen.

**Comments** If row contains a masked private database record, then the item remains unselected.

This function cannot highlight an entire row; it can only highlight one cell in a row, and it always unhighlights the previously selected table cell. If you want to select an entire row, either create a table that has a single column, or write your own selection code.

If the selected item is a multi-line text field or a text field with a nonstandard height, this function only highlights the top eleven pixels. If you want a larger area highlighted, you must write your own selection code.

**See Also** [TblRowSelectable](#), [TblGetItemBounds](#), [TblGetItemInt](#)

## TblSetBounds

<b>Purpose</b>	Sets the bounds of a table.				
<b>Declared In</b>	Table.h				
<b>Prototype</b>	<code>void TblSetBounds (TableType *tableP, const RectangleType *rP)</code>				
<b>Parameters</b>	<table><tr><td>-&gt; tableP</td><td>Pointer to a table object. (See <a href="#">TableType</a>.)</td></tr><tr><td>-&gt; rP</td><td>Pointer to a RectangleType structure that specifies the bounds for the table.</td></tr></table>	-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)	-> rP	Pointer to a RectangleType structure that specifies the bounds for the table.
-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)				
-> rP	Pointer to a RectangleType structure that specifies the bounds for the table.				
<b>Result</b>	Returns nothing.				
<b>Compatibility</b>	Implemented only if <a href="#">2.0 New Feature Set</a> is present.				

## TblSetColumnEditIndicator

<b>Purpose</b>	Set the column attribute that controls whether a column highlights when the table is in edit mode.						
<b>Declared In</b>	Table.h						
<b>Prototype</b>	<code>void TblSetColumnEditIndicator (TableType *tableP, Int16 column, Boolean editIndicator)</code>						
<b>Parameters</b>	<table><tr><td>-&gt; tableP</td><td>Pointer to a table object. (See <a href="#">TableType</a>.)</td></tr><tr><td>-&gt; column</td><td>Column number (zero based).</td></tr><tr><td>-&gt; editIndicator</td><td>true to highlight, false to turn off highlight.</td></tr></table>	-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)	-> column	Column number (zero based).	-> editIndicator	true to highlight, false to turn off highlight.
-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)						
-> column	Column number (zero based).						
-> editIndicator	true to highlight, false to turn off highlight.						
<b>Result</b>	Returns nothing.						
<b>Comments</b>	The edit indicator controls whether the item in the column is highlighted when it is selected. By default, text field items have the						

## Tables

### Table Functions

---

`editIndicator` value of `false`, while all other table item types have an edit indicator of `true`.

When the table is drawn, only the leftmost contiguous set of items with the edit indicator set are drawn as highlighted. That is, if columns 1, 2, and 4 have an edit indicator of `true` and column 3 has an edit indicator of `false`, only the items in column 1 and 2 are drawn as highlighted when selected. Column 4 items are not drawn as highlighted.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

**See Also** [TableColumnAttrType](#)

## TblSetColumnMasked

**Purpose** Set whether the column is masked.

**Declared In** Table.h

**Prototype** void TblSetColumnMasked (TableType \*tableP,  
Int16 column, Boolean masked)

**Parameters**

-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)
-> column	Column number (zero-based).
-> masked	<code>true</code> to have the column be masked, <code>false</code> otherwise.

**Result** Returns nothing.

**Comments** Masked cells are displayed as shaded. You should set a column to masked if its contents should be hidden when it contains information from a private database record and the user has set the display preference for private records to masked.

A table cell is not masked unless both its row and column are masked. This allows non-private information in the row item to remain visible. For example, the Datebook application shows the time for a private appointment, but it does not show the description.

Because the number of columns is static, you only need to call this function once per column when you first set up the table. The masked attribute on the row will determine if the contents of the table cell are actually displayed as shaded.

**Compatibility** Implemented only if [3.5 New Feature Set](#) if present.

**See Also** [TblRowMasked](#), [TblSetRowMasked](#), [TableColumnAttrType](#), [SecSelectViewStatus](#)

## TblSetColumnSpacing

**Purpose** Set the spacing after the specified column.

**Declared In** Table.h

**Prototype** void TblSetColumnSpacing (TableType \*tableP,  
Int16 column, Coord spacing)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)  
-> column Column number (zero-based).  
-> spacing Spacing after the column in pixels.

**Result** Returns nothing.

This function may display a fatal error message if the column parameter is invalid.

**See Also** [TblSetColumnUsable](#), [TableColumnAttrType](#)

## Tables

### Table Functions

---

## TblSetColumnUsable

**Purpose** Set a column in a table to usable or unusable.

**Declared In** Table.h

**Prototype** void TblSetColumnUsable (TableType \*tableP,  
Int16 column, Boolean usable)

**Parameters**

-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)
-> column	Column number (zero-based).
-> usable	true for usable or false for not usable.

**Result** Returns nothing.

This function may display a fatal error message if the column parameter is invalid.

**Comments** Columns that are not usable do not display.

**See Also** [TblMarkRowInvalid](#), [TableColumnAttrType](#)

## TblSetColumnWidth

**Purpose** Set the width of the specified column.

**Declared In** Table.h

**Prototype** void TblSetColumnWidth (TableType \*tableP,  
Int16 column, Coord width)

**Parameters**

-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)
-> column	Column number (zero-based).
-> width	Width of the column (in pixels).

**Result** Returns nothing.

This function may display a fatal error message if the column parameter is invalid.

**See Also** [TblGetColumnWidth](#), [TableColumnAttrType](#)

## TblSetCustomDrawProcedure

**Purpose** Set the custom draw callback procedure for the specified column.

**Declared In** Table.h

**Prototype** void TblSetCustomDrawProcedure  
(TableType \*tableP, Int16 column,  
TableDrawItemFuncPtr drawCallback)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)  
-> column Column number.  
-> drawCallback Callback function.

**Result** Returns nothing.

**Comments** The custom draw callback function is used to draw table items with a TableItemType of customTableItem or tallCustomTableItem. See the [TableItemType](#) description for more information.

This function may display a fatal error message if the column parameter is invalid.

**See Also** [TableDrawItemFuncType](#), [TblDrawTable](#),  
[TableColumnAttrType](#)

## Tables

### Table Functions

---

## TblSetItemFont

**Purpose** Set the font used to display a table item.

**Declared In** Table.h

**Prototype** void TblSetItemFont (TableType \*tableP,  
Int16 row, Int16 column, FontID fontID)

**Parameters**

-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)
-> row	Row number of the item (zero-based).
-> column	Column number of the item (zero-based).
-> fontID	ID of the font to be used.

**Result** Returns nothing.

**Comments** This function sets the value stored in the fontID field for this table item. Only certain types of table items use the font specified by the fontID field when they are displayed. The [TableItemType](#) description specifies what font is used to display each type of table item. It is not an error to set the fontID for a table item that does not use it.

This function may display a fatal error message if the row or column parameter specifies a row or column that is not on the screen.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [TblGetItemFont](#)

## TblSetItemInt

**Purpose** Set the integer value of the specified item.

**Declared In** Table.h

**Prototype** void TblSetItemInt (TableType \*tableP, Int16 row,  
Int16 column, Int16 value)

**Parameters**

-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)
-> row	Row number of the item (zero-based).
-> column	Column number of the item (zero-based).
-> value	Any byte value (an integer).

**Result** Returns nothing.

This function may display a fatal error message if the `row` or `column` parameter is invalid.

**Comments** You typically use this function when setting up and initializing a table for the first time to set the value of each table cell.

This function sets the value stored in the `intValue` field for this table item. Certain types of table items display the value stored in `intValue`, and other types display the value pointed to by the `ptr` field. See the [TableItemType](#) description for details. If you set the `intValue` of an item that displays its `ptr` value, it is not an error. An application can store whatever value it wants in the `intValue` field; however, be aware that this has nothing to do with the value displayed by such a table cell.

**See Also** [TblGetInt](#), [TblSetItemPtr](#)

## Tables

### Table Functions

---

## TblSetItemPtr

**Purpose** Set the item to the specified pointer value.

**Declared In** Table.h

**Prototype** void TblSetItemPtr (TableType \*tableP, Int16 row,  
Int16 column, void \*value)

**Parameters**

-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)
-> row	Row number of the item (zero-based).
-> column	Column number of the item (zero-based).
-> value	Pointer to data to display in the table item.

**Result** Returns nothing.

This function may display a fatal error message if the `row` or `column` parameter is invalid.

**Comments** This function sets the value stored in the `ptr` field for this table item. Certain types of table items display the value pointed to by `ptr`, and other types display the value stored in the `intValue` field. See the [TableItemType](#) description for details. If you set the `ptr` of an item that displays its `intValue`, it is not an error. An application can store whatever value it wants in the `ptr` field; however, be aware that this has nothing to do with the value displayed by such a table cell.

**See Also** [TblGetItemPtr](#), [TblSetItemInt](#)

## TblSetItemStyle

**Purpose** Set the type of item to display; for example, text, numbers, dates, and so on.

**Declared In** Table.h

**Prototype** void TblSetItemStyle (TableType \*tableP,  
Int16 row, Int16 column, TableItemType type)

## Tables

### Table Functions

---

<b>Parameters</b>	<code>-&gt; tableP</code>	Pointer to a table object. (See <a href="#">TableType</a> .)
	<code>-&gt; row</code>	Row number of the item (zero-based).
	<code>-&gt; column</code>	Column number of the item (zero-based).
	<code>-&gt; type</code>	The type of item, such as an editable text field or a check box. See <a href="#">TableItemType</a> for a list of possible values.
<b>Result</b>	Returns nothing. This function may display a fatal error message if the <code>row</code> or <code>column</code> parameter is invalid.	
<b>Comments</b>	<p>You typically use this function when first setting up and initializing a table; you do not dynamically change item styles.</p> <p>Follow this function with a call to either <a href="#">TblSetItemInt</a> or <a href="#">TblSetItemPtr</a> to set the value displayed by the table item. You should call one or the other of these functions depending on the type of table item you specified. See the table in the <a href="#">TableItemType</a> description for details.</p> <p>Note also that a table column always contains items of the same type. For example, you might initialize a table using this code:</p> <pre>for (row = 0; row &lt; rowsInTable; row++) {     TblSetItemStyle (table, row, completedColumn,                     checkboxTableItem);     TblSetItemStyle (table, row, priorityColumn,                     numericTableItem);     TblSetItemStyle (table, row, descColumn,                     textTableItem);     TblSetItemStyle (table, row, dueDateColumn,                     customTableItem);     TblSetItemStyle (table, row, categoryColumn,                     customTableItem); }</pre>	

**See Also** [TblSetCustomDrawProcedure](#)

## TblSetLoadDataProcedure

**Purpose** Set the load-data callback procedure for the specified column.

**Declared In** Table.h

**Prototype** void TblSetLoadDataProcedure (TableType \*tableP,  
Int16 column,  
TableLoadDataFuncPtr loadDataCallback)

**Parameters**

-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)
-> column	Column number (zero-based).
-> loadDataCallback	Callback procedure. See <a href="#">TableLoadDataFuncType</a> .

**Result** Returns nothing.

**Comments** The callback procedure is used to load the data values of a table item. See the [TableLoadDataFuncType](#) for more information on writing the callback function.

You typically use this function when first setting up and initializing a table.

**See Also** [TblSetCustomDrawProcedure](#)

## TblSetRowData

**Purpose** Set the data value of the specified row. The data value is a placeholder for application-specific values.

**Declared In** Table.h

**Prototype** void TblSetRowData (TableType \*tableP, Int16 row,  
UInt32 data)

**Parameters**

-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)
-----------	--

## Tables

### Table Functions

---

-> row	Row number (zero-based).
-> data	Application-specific data value to store for this row. For example, the Datebook and ToDo applications use this field to store the unique ID of the database record displayed by this row.

**Result** Returns nothing.

This function may display a fatal error message if the `row` parameter is invalid.

**See Also** [TblGetRowData](#), [TblFindRowData](#)

## TblSetRowHeight

**Purpose** Set the height of the specified row.

**Declared In** Table.h

**Prototype** void TblSetRowHeight (TableType \*tableP,  
Int16 row, Coord height)

<b>Parameters</b>	-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)
	-> row	Row number (zero-based).
	-> height	New height in pixels.

**Result** Returns nothing.

This function may display a fatal error message if the `row` parameter is invalid.

**See Also** [TblGetRowHeight](#), [TblSetRowStaticHeight](#)

## TblSetRowID

**Purpose** Set the ID value of the specified row.

**Declared In** Table.h

**Prototype** void TblSetRowID (TableType \*tableP, Int16 row,  
                  UInt16 id)

## Tables

### Table Functions

---

<b>Parameters</b>	-> tableP -> row -> id	Pointer to a table object. (See <a href="#">TableType</a> .) Row number (zero-based). ID to identify a row.
-------------------	------------------------------	---

<b>Result</b>	Returns nothing.  This function may display a fatal error message if the <code>row</code> parameter is invalid.
---------------	---

**See Also** [TblGetRowID](#), [TblFindRowID](#), [TableRowAttrType](#)

## TblSetRowMasked

<b>Purpose</b>	Set a row in a table to masked or unmasked.
----------------	---

<b>Declared In</b>	Table.h
--------------------	---------

<b>Prototype</b>	<code>void TblSetRowMasked (TableType *tableP, Int16 row, Boolean masked)</code>
------------------	--

<b>Parameters</b>	-> tableP -> row -> masked	Pointer to a table object. (See <a href="#">TableType</a> .) Row number (zero-based). <code>true</code> to have the row be masked, <code>false</code> otherwise.
-------------------	----------------------------------	--

<b>Result</b>	Returns nothing.
---------------	------------------

<b>Comments</b>	Masked cells are displayed as shaded. You should call this function before drawing or redrawing the table. If a table row contains a private database record and the user has set the display preference for private records to masked, you must call this function on that row. For example:
-----------------	---

```
UInt16 attr;  
privateRecordViewEnum privateRecordStatus;  
Boolean masked;  
  
privateRecordStatus = (privateRecordViewEnum)
```

```
PrefGetPreference(prefShowPrivateRecords) ;  
....  
DmRecordInfo (ToDoDB, recordNum, &attr, NULL,  
NULL) ;  
masked = ((attr & dmRecAttrSecret) &&  
(privateRecordStatus == maskPrivateRecords)) ;  
TblSetRowMasked(tableP, row, masked) ;
```

Note that a table cell is not masked unless both its row and column are masked. This allows non-private information in the row item to remain visible. For example, the Datebook application shows the time for a private appointment, but it does not show the description.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [TblRowMasked](#), [TblSetColumnMasked](#), [TableRowAttrType](#), [SecSelectViewStatus](#)

## TblSetRowSelectable

**Purpose** Set a row in a table to selectable or nonselectable.

**Declared In** Table.h

**Prototype** void TblSetRowSelectable (TableType \*tableP,  
Int16 row, Boolean selectable)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)  
-> row Row number (zero-based).  
-> selectable true or false.

**Result** Returns nothing.

This function may display a fatal error message if the `row` parameter is invalid.

**Comments** Rows that are not selectable don't highlight when touched.

**See Also** [TblRowSelectable](#), [TblSetRowUsable](#), [TableRowAttrType](#)

## Tables

### Table Functions

---

## TblSetRowStaticHeight

<b>Purpose</b>	Set the static height attribute of a row.						
<b>Declared In</b>	Table.h						
<b>Prototype</b>	<code>void TblSetRowStaticHeight (TableType *tableP, Int16 row, Boolean staticHeight)</code>						
<b>Parameters</b>	<table><tr><td>-&gt; tableP</td><td>Pointer to a table object. (See <a href="#">TableType</a>.)</td></tr><tr><td>-&gt; row</td><td>Row number (zero-based).</td></tr><tr><td>-&gt; staticHeight</td><td>true to set the static height, false to unset it.</td></tr></table>	-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)	-> row	Row number (zero-based).	-> staticHeight	true to set the static height, false to unset it.
-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)						
-> row	Row number (zero-based).						
-> staticHeight	true to set the static height, false to unset it.						
<b>Result</b>	Nothing.  This function may display a fatal error message if the <code>row</code> parameter is invalid.						
<b>Comments</b>	A row that has its static height attribute set will not expand or contract the height of the row as text is added or removed from a text item.						
<b>Compatibility</b>	Implemented only if <a href="#">2.0 New Feature Set</a> is present.						

## TblSetRowUsable

<b>Purpose</b>	Set a row in a table to usable or unusable. Rows that are not usable do not display.				
<b>Declared In</b>	Table.h				
<b>Prototype</b>	<code>void TblSetRowUsable (TableType *tableP, Int16 row, Boolean usable)</code>				
<b>Parameters</b>	<table><tr><td>-&gt; tableP</td><td>Pointer to a table object. (See <a href="#">TableType</a>.)</td></tr><tr><td>-&gt; row</td><td>Row number (zero-based).</td></tr></table>	-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)	-> row	Row number (zero-based).
-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)				
-> row	Row number (zero-based).				

-> usable              true or false.

**Result** Returns nothing.

This function may display a fatal error message if the `row` parameter is invalid.

**See Also** [TblRowUsable](#), [TblSetRowSelectable](#)

## TblSetSaveDataProcedure

**Purpose** Set the save-data callback procedure for the specified column.

**Declared In** Table.h

**Prototype** void TblSetSaveDataProcedure (TableType \*tableP,  
                  Int16 column,  
                  TableSaveDataFuncPtr saveDataCallback)

**Parameters** -> `tableP`              Pointer to a table object. (See [TableType](#).)  
                  -> `column`              Column number (zero-based).  
                  -> `saveDataCallback`  
                                    Callback function. See  
                                    [TableSaveDataFuncType](#).

**Result** Returns nothing.

This function may display a fatal error message if the `column` parameter is invalid.

**Comments** The callback procedure is called when the table object determines the data of a text object needs to be saved.

**See Also** [TblSetCustomDrawProcedure](#)

## Tables

### Table Functions

---

## TblSetSelection

<b>Purpose</b>	Set a table item.						
<b>Declared In</b>	Table.h						
<b>Prototype</b>	<code>void TblSetSelection (TableType *tableP, Int16 row, Int16 column)</code>						
<b>Parameters</b>	<table><tr><td><code>-&gt; tableP</code></td><td>Pointer to a <a href="#">TableType</a>.</td></tr><tr><td><code>-&gt; row</code></td><td>Table row.</td></tr><tr><td><code>-&gt; column</code></td><td>Table column.</td></tr></table>	<code>-&gt; tableP</code>	Pointer to a <a href="#">TableType</a> .	<code>-&gt; row</code>	Table row.	<code>-&gt; column</code>	Table column.
<code>-&gt; tableP</code>	Pointer to a <a href="#">TableType</a> .						
<code>-&gt; row</code>	Table row.						
<code>-&gt; column</code>	Table column.						
<b>Result</b>	Returns nothing.						
<b>Comments</b>	This function sets a table item, determined by the row and column arguments, as the current selection. <a href="#">TblDrawTable</a> must be called afterwards to update the UI.						
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call <code>TblGlueSetSelection</code> . For more information, see <a href="#">Chapter 75, “PalmOSGlue Library.”</a>						
<b>See Also</b>	<a href="#">TblGetNumberOfColumns</a> , <a href="#">TblGetTopRow</a>						

## TblUnhighlightSelection

**Purpose** Unhighlight the currently selected item in a table.

**Declared In** Table.h

**Prototype** void TblUnhighlightSelection (TableType \*tableP)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)

**Result** Returns nothing.

## Application-Defined Functions

### TableDrawItemFuncType

**Purpose** Draw a custom table item.

**Declared In** Table.h

**Prototype** void TableDrawItemFuncType (void \*tableP,  
Int16 row, Int16 column, RectangleType \*bounds)

**Parameters** -> tableP Pointer to a table object. (See [TableType](#).)  
-> row Row number of the item to be drawn (zero-based).  
-> column Column number of the item to be drawn (zero-based).  
-> bounds The area of the screen in which the item is to be drawn.

**Result** Returns nothing.

**Comments** This function is called during [TblDrawTable](#) and [TblRedrawTable](#).

## Tables

### *Application-Defined Functions*

---

You implement a custom drawing function when your table contains items of type `customTableItem` or `tallCustomTableItem` (to draw the entire item) or `narrowTextTableItem` (to draw whatever is required in the space between the text and the right edge of the table cell).

You may implement a custom drawing function to include any style of information in the table. If you don't like the way a predefined item is drawn, you may prefer to use a `customTableItem` or `tallCustomTableItem` instead. For example, if you want to include a date in your table but you want it to show the year as well as the month and day, you should implement a custom drawing function.

**See Also** [TblSetCustomDrawProcedure](#), [TableItemType](#)

## TableLoadDataFuncType

**Purpose** Load data into a column.

**Declared In** Table.h

**Prototype** Err TableLoadDataFuncType (void \*tableP,  
Int16 row, Int16 column, Boolean editable,  
MemHandle \*dataH, Int16 \*dataOffset,  
Int16 \*dataSize, FieldPtr fld)

<b>Parameters</b>	-> tableP	Pointer to a table object. (See <a href="#">TableType</a> .)
	-> row	Row number of the table item to load.
	-> column	Column number of the table item to load.
	-> editable	If true, the table is currently being edited. If false, the table is being drawn but not necessarily being edited.
	<- dataH	Unlocked handle of a block containing a null-terminated text string.
	<- dataOffset	Offset from start of block to start of the text string.

<- dataSize      Allocated size of text string, **not** the string length.

-> fld            Pointer to the text field in this table cell.

**Result**    Returns 0 upon success or an error if unsuccessful.

**Comments**    This function is called in two cases: when a text field item is being drawn ([TblDrawTable](#) or [TblRedrawTable](#)) and when a text field item is being selected (part of [TblHandleEvent](#)'s handling of [tblEnterEvent](#)). If this function returns an error (any nonzero value) and the item is being selected, then the item is not selected and the table's editing attribute is set to false.

You return the same values for dataH, dataOffset, and dataSize that you would pass to [FldSetText](#). That is, you can use this function to point the table cell's text field to a string in a database record so that you can edit that string directly using text field routines. To do so, return the handle to a database record in dataH, the offset from the start of the record to the start of the string in dataOffset, and the allocated size of the string in dataSize.

The handle that you return from this function is assumed to contain a null-terminated string starting at dataOffset bytes in the memory chunk. The string should be between 0 and dataSize - 1 bytes in length.

As with [FldSetText](#), you are responsible for freeing the memory associated with the dataH parameter. You can do so in the [TableSaveDataFuncType](#) function, but it is only called for a cell that has been edited. For non-editable text cells or text cells that are editable but were never selected, free the memory when you close the form.

The fld pointer passed to your function has already been initialized with default values by the table code. If you want to override a field's attributes (for example, if you want to change the underline mode) you can do so in this function.

**See Also**    [TblDrawTable](#), [TblHandleEvent](#), [TableLoadDataFuncType](#)

## Tables

### *Application-Defined Functions*

---

## TableSaveDataFuncType

<b>Purpose</b>	Save the data associated with a text field.
<b>Declared In</b>	Table.h
<b>Prototype</b>	Boolean TableSaveDataFuncType (void *tableP, Int16 row, Int16 column)
<b>Parameters</b>	-> tableP              Pointer to a table object. (See <a href="#">TableType</a> .) -> row                  Row number of the table item to load. -> column               Column number of the table item to load.
<b>Result</b>	Return <code>true</code> if the table should be redrawn, or <code>false</code> if the table does not need to be redrawn.
<b>Comments</b>	This is called before the memory associated with the currently selected text field in a table cell is freed. Implement this function if you need to do any special cleanup before this memory is freed.  This function is called only when the currently selected editable text field is releasing the focus. You can use <a href="#">TblGetCurrentField</a> to retrieve a pointer to this field. It is called only on the currently selected field, not on any other fields in the table.  Note that the table manager already disassociates the memory handle from the text field for you so that the memory associated with your data is not freed when the field is freed. The table manager also calls <a href="#">FldCompactText</a> for you.  If the text handle you returned in your <a href="#">TableLoadDataFuncType</a> callback points to a string on the dynamic heap, you should implement this callback function to store or free the handle. You can use <a href="#">FldGetTextHandle</a> to obtain the handle.  If you return <code>true</code> from this function, <a href="#">TblRedrawTable</a> is called. You should mark invalid any table rows that you want redrawn before returning.
<b>See Also</b>	<a href="#">TblSetSaveDataProcedure</a>

# UI Color List

---

This chapter provides information about the UI color list by discussing the following topics:

- [UI Color Data Types](#)
- [UI Color Functions](#)

The header file `UIColor.h` declares the API that this chapter describes. For more information on the color list, see “[Color and Grayscale Support](#)” on page 144 in the *Palm OS Programmer’s Companion*, vol. I.

## UI Color Data Types

### **UIColorTableEntries**

The `UIColorTableEntries` enum declares symbolic color constants for the various UI elements.

Do not confuse the UI color list with the system color table. The **system color table** (or **system palette**) defines all available colors for the display or draw window, whether they are in use or not. The **UI color list** defines the colors used to draw the interface objects.

```
typedef enum UIColorTableEntries {
    UIObjectFrame = 0,
    UIObjectFill,
    UIObjectForeground,
    UIObjectSelectedFill,
    UIObjectSelectedForeground,
    UIMenuFrame,
    UIMenuFill,
    UIMenuForeground,
    UIMenuSelectedFill,
    UIMenuSelectedForeground,
```

## UI Color List

### *UI Color Data Types*

---

```
UIFieldBackground,  
UIFieldText,  
UIFieldTextLines,  
UIFieldCaret,  
UIFieldTextHighlightBackground,  
UIFieldTextHighlightForeground,  
UIFieldFepRawText,  
UIFieldFepRawBackground,  
UIFieldFepConvertedText,  
UIFieldFepConvertedBackground,  
UIFieldFepUnderline,  
  
UIFormFrame,  
UIFormFill,  
  
UIDialogFrame,  
UIDialogFill,  
  
UIAlertFrame,  
UIAlertFill,  
  
UIOK,  
UICaution,  
UIWarning,  
  
UILastColorTableEntry  
} UIColorTableEntries;
```

## Field Descriptions

UIObjectFrame	Color for the border of user interface objects (such as command buttons, push buttons, selector triggers, menus, arrows checkboxes, and other controls).
UIObjectFill	The background color for a solid or “filled” user interface object.  Note that UI objects in tables use the <code>UIField...</code> colors instead of the <code>UIObject...</code> colors.
UIObjectForeground	The color for foreground items (such as labels or graphics) in a user interface object.
UIObjectSelectedFill	The background color of the currently selected user interface object, whether that object is solid or not.
UIObjectSelectedForeground	The color of foreground items in a selected user interface object.
UIMenuFrame	The color of the border around the menu.
UIMenuFill	The background color of a menu item.
UIMenuForeground	The color of the menu’s text.
UIMenuSelectedFill	The background color of a selected menu item.
UIMenuSelectedForeground	The color of the text of a selected menu item.
UIFieldBackground	The background color of an editable text field.
UIFieldText	The color of the text in the editable field.
UIFieldTextLines	The color of underlines in an editable field.
UIFieldCaret	The color of the cursor in an editable text field.
UIFieldTextHighlightBackground	The background color for selected text in an editable text field.

## UI Color List

### *UI Color Data Types*

---

UITextFieldHighlightForeground	The color of the selected text in an editable text field.
UITextFieldFepRawText	Color used for unconverted text in the inline conversion area when a FEP is used as a text input method (for example, on Japanese devices).  If the FEP colors are identical to field colors, unconverted text has a solid underline.
UITextFieldFepRawBackground	The background color for unconverted text in the inline conversion area when a FEP is used as a text input method.  If the FEP colors are identical to field colors, unconverted text has a solid underline.
UITextFieldFepConvertedText	Color used for converted text in the inline conversion area when a FEP is used as a text input method (for example, on Japanese devices).  If the FEP colors are identical to field colors, converted text has a double-thick underline.
UITextFieldFepConvertedBackground	The background color used for converted text in the inline conversion area.  If the FEP colors are identical to field colors, converted text has a double-thick underline.
UITextFieldFepUnderline	The color used for underlines in the inline conversion area.
UIFormFrame	The color of the border and titlebar on a form.
UIFormFill	The background color of a form. White is recommended for this value.
UIDialogFrame	The color of a border and titlebar on a modal form.

UIDialogFill	The background color of a modal form.
UIAlertFrame	The color of the border and titlebar on an alert panel.
UIAlertFill	The background color of an alert panel.
UIOK	The color for an informational icon.
UICaution	The color for a caution icon.
UIWarning	The color for a warning icon.
UILastColorTableEntry	Palm OS® does not currently use the UIOK, UICaution, and UIWarning constants. Placeholder to indicate end of enum.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

## UI Color Functions

### UIColorGetTableEntryIndex

<b>Purpose</b>	Return the index value for a UI color for the current system palette.	
<b>Declared In</b>	UIColor.h	
<b>Prototype</b>	IndexedColorType	UIColorGetTableEntryIndex (UIColorTableEntries which)
<b>Parameters</b>	-> which	One of the symbolic color constants. See <a href="#">UIColorTableEntries</a> .
<b>Result</b>	Returns the system color table index of the color used for the specified symbolic color.	
<b>Comments</b>	One way to find out the indexes of all the colors that the OS is using is to loop through the UI color list, calling	

## **UI Color List**

### *UI Color Functions*

---

`UIColorGetTableEntryIndex` for each slot, and keep a list (excluding duplicates).

```
IndexedColorType  
    colorsUsed[UILastColorTableEntry] ;  
UInt16 numColors = 0;  
...  
for (i = 0; i < UILastColorTableEntry; i++) {  
    IndexedColorType currentColor;  
    Boolean isNew = true;  
  
    currentColor = UIColorGetTableEntryIndex(i);  
  
    for (j = 0; ((j < numColors) && isNew); j++)  
        if (colorsUsed[j] == currentColor)  
            isNew = false; /* exit loop */  
    if (isNew) {  
        numColors++;  
        colorsUsed[j] = currentColor;  
    }  
}
```

To get the RGB values of the colors, do the same thing but call [UIColorGetTableEntryRGB](#).

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [IndexedColorType](#), [WinIndexToRGB](#)

## UIColorGetTableEntryRGB

**Purpose** Return the RGB value for the UI color.

**Declared In** UIColor.h

**Prototype** void UIColorGetTableEntryRGB  
(UIColorTableEntries which, RGBColorType \*rgbP)

**Parameters** -> which One of the symbolic color constants. See [UIColorTableEntries](#).

<- rgbP Pointer to an RGB color value corresponding to the current color used for the symbolic color. (See [RGBColorType](#).)

## **UI Color List**

### *UI Color Functions*

---

<b>Result</b>	Returns nothing.
<b>Comments</b>	In general, it is more efficient to work with indexed color entries instead of RGB color entries.
<b>Compatibility</b>	Implemented only if <a href="#">3.5 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">UIColorGetTableEntryIndex</a> , <a href="#">WinRGBToIndex</a>

## **UIColorSetTableEntry**

<b>Purpose</b>	Change a value in the UI color list.				
<b>Declared In</b>	<code>UIColor.h</code>				
<b>Prototype</b>	<code>Err UIColorSetTableEntry (UIColorTableEntries which, const RGBColorType *rgbP)</code>				
<b>Parameters</b>	<table><tr><td>-&gt; which</td><td>One of the symbolic color constants. See <a href="#">UIColorTableEntries</a>.</td></tr><tr><td>-&gt; rgbP</td><td>The RGB value of the color that should be used for the specified UI object. (See <a href="#">RGBColorType</a>.)</td></tr></table>	-> which	One of the symbolic color constants. See <a href="#">UIColorTableEntries</a> .	-> rgbP	The RGB value of the color that should be used for the specified UI object. (See <a href="#">RGBColorType</a> .)
-> which	One of the symbolic color constants. See <a href="#">UIColorTableEntries</a> .				
-> rgbP	The RGB value of the color that should be used for the specified UI object. (See <a href="#">RGBColorType</a> .)				
<b>Result</b>	Returns 0 upon success.				
<b>Comments</b>	Sets the value of a UI color entry to the passed RGB value. Updates the index for that UI color entry to the current best fit for that RGB value according to the palette used by the current draw window.  It is best to use this function only if the draw window is currently onscreen. Otherwise, the best-fit algorithm may choose a color that is not available on the current screen.				
<b>See Also</b>	<a href="#">WinIndexToRGB</a> , <a href="#">UIColorGetTableEntryIndex</a> , <a href="#">UIColorGetTableEntryRGB</a>				

## **UI Color List**

### *UI Color Functions*

---

# UI Controls

---

This chapter describes the UI controls API as declared in `UIControls.h`.

## UI Control Functions

### **UIBrightnessAdjust**

**Purpose** Displays the brightness adjust dialog.

**Declared In** `UIControls.h`

**Prototype** `void UIBrightnessAdjust()`

**Parameters** None

**Result** Returns nothing.

**Comments** On hardware that supports a brightness setting, this function displays a dialog that allows the user to change the brightness level. On hardware that has a backlight, this function toggles the backlight.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

## UIContrastAdjust

<b>Purpose</b>	Displays the contrast adjust dialog (currently only available on the Palm V™ Connected Organizer).
<b>Declared In</b>	UIControls.h
<b>Prototype</b>	void UIContrastAdjust()
<b>Parameters</b>	None.
<b>Result</b>	Returns nothing.
<b>Compatibility</b>	This function was renamed from ContrastAdjust to UIContrastAdjust in Palm OS® release 3.5. The ContrastAdjust function is available if <a href="#">3.1 New Feature Set</a> is present.

## UIPickColor

<b>Purpose</b>	Displays a dialog that allows the user to choose a color.	
<b>Declared In</b>	UIControls.h	
<b>Prototype</b>	Boolean UIPickColor (IndexedColorType *indexP, RGBColorType *rgbP, UIPickColorStartType start, const Char *titleP, const Char *tipP)	
<b>Parameters</b>	<-> indexP	Index value of the selected color. (See <a href="#">IndexedColorType</a> .) Upon entry, this points to the index value of the color initially selected. Upon return, this points to the index value of the color the user selected. Pass NULL to not set or return this value.

<-> rgbP	RGB value of the selected color. (See <a href="#">RGBColorType</a> .) Upon entry, this points to the RGB value of the color initially selected when the dialog is displayed. Upon return, this points to the RGB value that the user selected. Pass NULL to not set or return this value.
-> start	Either UIPickColorStartPalette to display the system palette as a series of color squares or UIPickColorStartRGB to display individual sliders for the red, green, and blue values. This parameter is only used if both indexP and rgbP are not NULL.
-> titleP	String to display as the title of the dialog. Specify NULL to use the default title, which is "Pick Color."
-> tipP	Not used.

**Result** Returns true if a new color was selected, false otherwise.

**Comments** Use this function to allow users to choose a color used in your user interface. (The system never calls UIPickColor.)

This function can display two versions of the dialog: palette or RGB. The palette version of the dialog displays a series of squares, each containing a different color defined on the system palette. The indexP value contains the index of the square that is initially selected.

The RGB version of the dialog displays three sliders that allow the user to select the level of red, green, and blue in the color. The rgbP parameter contains the red, green, and blue values initially shown in the dialog. The sliders only allow values that are defined in the current system color table.

If indexP is initially NULL, only the RGB dialog is displayed. Similarly, if rgbP is NULL, only the palette version is displayed. If both parameters are non-NULL, the system adds a pull-down list that allows the user to switch between the palette dialog and the RGB dialog, and the start parameter controls which version of the

dialog is initially shown. In this case, both `indexP` and `rgbP` contain the value of the user-selected color upon return.

Palm OS 3.5 supports a maximum of 256 colors. The number of possible RGB colors greatly exceeds this amount. For this reason, the chosen RGB may not have an exact match. If this is the case, the `indexP` parameter (if not NULL) contains the closest match using a luminance best-fit if the color lookup table is entirely grayscale (red, green, and blue values for each entry are identical), or a shortest-distance fit in the RGB space is used if the palette contains colors.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [WinSetBackColor](#), [WinSetForeColor](#), [WinSetTextColor](#),  
[UIColorSetTableEntry](#)

# Miscellaneous User Interface Functions

---

This chapter provides descriptions of miscellaneous user interface functions. It covers the following topics:

- [Miscellaneous User Interface Data Structures](#)
- [Miscellaneous User Interface Functions](#)

You can find declarations for the functions described in this chapter in the header files AppLaunchCmd.h, PhoneLookup.h, and UIResources.h.

## Miscellaneous User Interface Data Structures

The [PhoneNumberLookupCustom](#) function uses these data structures to look up contact information based upon the current cursor position.

### AddressLookupFields

The AddressLookupFields enum specifies the fields you can search by and the corresponding fields to return using the field1 and field2 elements of the [AddrLookupParamsType](#) structure. For both field1 and field2 pass one of the values up to, but not including, addrLookupFieldCount.

```
typedef enum {
    addrLookupName,
    addrLookupFirstName,
    addrLookupCompany,
    addrLookupAddress,
    addrLookupCity,
    addrLookupState,
    addrLookupZipCode,
```

## Miscellaneous User Interface Functions

### *Miscellaneous User Interface Data Structures*

---

```
    addrLookupCountry,
    addrLookupTitle,
    addrLookupCustom1,
    addrLookupCustom2,
    addrLookupCustom3,
    addrLookupCustom4,
    addrLookupNote,
    addrLookupWork,
    addrLookupHome,
    addrLookupFax,
    addrLookupOther,
    addrLookupEmail,
    addrLookupMain,
    addrLookupPager,
    addrLookupMobile,
    addrLookupSortField,
    addrLookupListPhone,
    addrLookupFieldCount,

    addrLookupNoField = 0xff
} AddressLookupFields;
```

## **AddrLookupParamsType**

Pass this structure to [PhoneNumberLookupCustom](#) to precisely control the phone number lookup dialog and paste process.

```
typedef struct {
    Char *title;
    Char *pasteButtonText;
    Char lookupString [addrLookupStringLength];
    AddressLookupFields field1;
    AddressLookupFields field2;
    Boolean field2Optional;
    Boolean userShouldInteract;
    Char *formatStringP;
    MemHandle resultStringH;
    UInt32 uniqueID;
} AddrLookupParamsType;
```

```
typedef AddrLookupParamsType  
*AddrLookupParamsPtr;
```

### Value Descriptions

title	Title to appear in the title bar. Supply NULL to use the default title.
pasteButtonText	Text to appear in paste button. Supply NULL to use the default, “paste”.
lookupString	Buffer containing the string to look up. If the string matches only one record, that record is used without presenting the lookup dialog to the user. <a href="#">PhoneNumberLookup</a> and <a href="#">PhoneNumberLookupCustom</a> both set this field based upon the current selection or cursor position.
field1	Field to search by. This field appears on the left side of the lookup dialog. If the field is the sort field, the search is performed using a binary search. If the field isn’t the sort field, searching is performed by a linear search, which can be slow. Supply one of the values in the <a href="#">AddressLookupFields</a> enum to specify the field to search by.
field2	Field to display on the right. Often displays some information about the person. If it is a phone field and a record has multiple instances of the phone type then the person appears once per instance of the phone type. Either field1 or field2 may be a phone field, but not both. Supply one of the values in the <a href="#">AddressLookupFields</a> enum to specify the field to display.

## Miscellaneous User Interface Functions

### *Miscellaneous User Interface Data Structures*

---

field2Optional	A value of <code>true</code> means that the record need not have <code>field2</code> in order to be listed. A value of <code>false</code> indicates that <code>field2</code> is required in order for the record to be listed.
userShouldInteract	A value of <code>true</code> forces the user to resolve non-unique lookups. A <code>false</code> value means a non-unique and complete lookup causes <code>resultStringH</code> to be set to <code>NULL</code> and <code>uniqueID</code> to be set to 0.

#### `formatStringP`

Controls the format of the paste string. All characters in the format string are literal unless they identify a field (signified by a caret (^) followed by the field name). For example, the format string “^first - ^home” might result in “Roger - 123-4567”. Allowable field names are:

- name
- first
- company
- address
- city
- state
- zipcode
- country
- title
- custom1
- custom2
- custom3
- custom4
- work
- home
- fax
- other
- email
- main
- pager
- mobile
- listname

## Miscellaneous User Interface Functions

### *Miscellaneous User Interface Functions*

---

resultStringH	If there is a format string, a result string is allocated on the dynamic heap and its handle is returned here.
uniqueID	The unique ID of the found record, or 0 if none was found, is returned here.

## Miscellaneous User Interface Functions

### PhoneNumberLookup

**Purpose** Calls the Address Book application to look up a phone number.

**Declared In** PhoneLookup.h

**Prototype** void PhoneNumberLookup (FieldType \*fldP)

**Parameters** -> fldP Field object in which the text to match is found.

**Result** Nothing returned; it's locked.

**Comments** This function displays the user's phone list and inserts the chosen name and number (or company name, name, and number, if that's how the user's Address Book preferences indicate that the phone list should be sorted) into the specified field. When displaying the phone list, PhoneNumberLookup scrolls the list to that entry that best matches the supplied field. The match compares the field contents against the name or company name (depending on the user's preferences) as follows:

- If the field contains selected text, PhoneNumberLookup tries to match against the selected text. The selected text is then replaced with the text of the chosen address list entry.
- If there is no selected text in the field, PhoneNumberLookup matches against the word in which the cursor lies (the match will take place if the cursor is at the beginning, the end, or within a word). The matched word is replaced with the text of the chosen address list entry.

- If the cursor does not lie within or adjoin a word, `PhoneNumberLookup` displays the address list starting at the first entry, and the text of the chosen entry is inserted at the current position within the text field.

If the user chooses Cancel when the address list is displayed, the field contents are left unaltered. The paste operation takes place through the clipboard so that Undo can be used to restore the field to its previous state.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

**See Also** [PhoneNumberLookupCustom](#)

## PhoneNumberLookupCustom

**Purpose** Calls the Address Book application to look up a phone number.

**Declared In** `PhoneLookup.h`

**Prototype** `void PhoneNumberLookupCustom (FieldType *fldP,  
AddrLookupParamsType *params,  
Boolean useClipboard)`

**Parameters** `-> fldP` Field object in which the text to match is found.

`<-> params` A structure that allows full control over the find dialog and the format of the resulting paste string. See [AddrLookupParamsType](#) for a description of the fields in this structure.

`-> useClipboard`

If true, `PhoneNumberLookupCustom` pastes the result into the field through the clipboard, thereby enabling undo.

**Result** Nothing returned; it's locked.

## Miscellaneous User Interface Functions

### *Miscellaneous User Interface Functions*

---

- Comments** This function displays two fields from each record in the user's address list and inserts a formatted string based upon fields in the chosen record into the specified field. When displaying the address list, `PhoneNumberLookupCustom` scrolls the list to that entry that best matches the supplied field. The match compares the field contents against the field specified in the `params` structure's `field1` element as follows:
- If the field contains selected text, `PhoneNumberLookup` tries to match against the selected text. The selected text is then replaced with the text of the chosen address list entry.
  - If there is no selected text in the field, `PhoneNumberLookup` matches against the word in which the cursor lies (the match will take place if the cursor is at the beginning, the end, or within a word). The matched word is replaced with the text of the chosen address list entry.
  - If the cursor does not lie within or adjoin a word, `PhoneNumberLookup` displays the address list starting at the first entry, and the text of the chosen entry is inserted at the current position within the text field.
- `PhoneNumberLookupCustom` copies the portion of the field used for the search—the selected text or the word in which the cursor lies—into the `lookupString` field in the `params` structure prior to replacing that part of the field with the user-selected entry.
- If the user chooses Cancel when the address list is displayed, the field contents are left unaltered. Depending on the value of the `useClipboard` parameter, the paste operation can take place through the clipboard so that Undo can be used to restore the field to its previous state.

- Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## ResLoadConstant

<b>Purpose</b>	Load a constant from a 'tint' resource and return its value.
<b>Declared In</b>	UIResources.h
<b>Prototype</b>	UInt32 ResLoadConstant (UInt16 rscID)
<b>Parameters</b>	-> rscID    The ID of the 'tint' resource (symbolically named constantRscType) to load.
<b>Result</b>	The four-byte value of the constant in the resource, or 0 if the resource could not be found. The return value may be cast as necessary.
<b>Comments</b>	<p>Use this function to load constant values that are stored as 'tint' resources. (All open resource databases are searched for the resource ID you specify.) You should store a constant value as a resource when its value changes depending on the locale.</p> <p>As an example, consider the maximum length of the Alarm Sound trigger label in the Datebook application's preferences panel. The list displayed by this trigger uses the localized name for each sound stored in the system. Because localized names are used, the maximum length that the Datebook application allows for the label differs depending on the current locale. The maximum length is stored as a resource constant so that each overlay database can specify a different value for the constant.</p>
<b>Compatibility</b>	Implemented only if <a href="#">3.5 New Feature Set</a> is present. To use this function in code intended to be run on earlier versions of Palm OS®, link with the PalmOSGlue library and call ResGlueLoadConstant. For more information, see <a href="#">Chapter 75, "PalmOSGlue Library."</a>
<b>See Also</b>	<a href="#">DmGetResource</a> , <a href="#">DmGet1Resource</a>

## Miscellaneous User Interface Functions

### *Miscellaneous User Interface Functions*

---

## ResLoadForm

<b>Purpose</b>	Copy and initialize a form resource. The structures are complete except pointers updating. Pointers are stored as offsets from the beginning of the form.
<b>Declared In</b>	UIResources.h
<b>Prototype</b>	<code>void* ResLoadForm (UInt16 rscID)</code>
<b>Parameters</b>	<code>-&gt; rscID</code> The resource ID of the form.
<b>Result</b>	The handle of the memory block that the form is in, since the form structure begins with the <a href="#">WindowType</a> , this is also a WinHandle.
<b>Compatibility</b>	If <a href="#">5.0 New Feature Set</a> is present this function is unimplemented.

## ResLoadMenu

<b>Purpose</b>	Copy and initialize a menu resource. The structures are complete except pointers updating. Pointers are stored as offsets from the beginning of the menu.
<b>Declared In</b>	UIResources.h
<b>Prototype</b>	<code>void* ResLoadMenu (UInt16 rscID)</code>
<b>Parameters</b>	<code>-&gt; rscID</code> The resource ID of the menu.

# **Part II: System Management**



# Alarm Manager

---

This chapter provides reference material for the alarm manager:

- [Alarm Manager Functions](#)
- [Application-Defined Functions](#)

The alarm manager API is declared in the header file `AlarmMgr.h`.

For more information on the Alarm Manager, see the section “[Alarms](#)” in the *Palm OS Programmer’s Companion*, vol. I.

## Alarm Manager Functions

### **AlmGetAlarm**

**Purpose**      Return the date and time for the application’s alarm, if one is set.

**Declared In**    `AlarmMgr.h`

**Prototype**     `UInt32 AlmGetAlarm (UInt16 cardNo, LocalID dbID,  
                      UInt32* refP)`

**Parameters**

-> <code>cardNo</code>	Number of the storage card on which the application resides.
-> <code>dbID</code>	Local ID of the application.

## Alarm Manager

### Alarm Manager Functions

---

<- refP      The alarm's reference value is returned here. This value is passed to the application when the alarm is triggered.

**Result**      The date and time the alarm will trigger, given in seconds since 1/1/1904; if no alarm is active for the application, 0 is returned for the alarm seconds and the reference value is undefined.

**See Also**      [AlmSetAlarm](#)

## AlmGetProcAlarm

**Purpose**      Macro that returns the date and time that a procedure alarm will trigger. Also returns the caller-defined alarm reference value.

**Declared In**      AlarmMgr.h

**Prototype**      AlmGetProcAlarm (procP, refP)

**Parameters**      -> procP      Pointer to a function that will be called when alarm is triggered. See [AlmAlarmProcPtr](#).  
                  <- refP      A UInt32 pointer to a location where the alarm's reference value is returned. This value is passed to the procedure when the alarm is triggered.

**Result**      The date and time the alarm will trigger, given in seconds since 1/1/1904; if no alarm is active for the procedure, 0 is returned for the alarm seconds and the reference value is undefined.

**Compatibility**      Implemented only if [3.2 New Feature Set](#) is present.

**See Also**      [AlmSetProcAlarm](#)

## AlmSetAlarm

**Purpose** Set or cancel an alarm for the given application.

**Declared In** AlarmMgr.h

**Prototype** Err AlmSetAlarm (UInt16 cardNo, LocalID dbID,  
UInt32 ref, UInt32 alarmSeconds, Boolean quiet)

<b>Parameters</b>	-> cardNo	Number of the storage card on which the application resides.
	-> dbID	Local ID of the application.
	-> ref	Caller-defined value. This value is passed with the launch code that notifies the application that the alarm has been triggered.
	-> alarmSeconds	Alarm date/time in seconds since 1/1/1904, or 0 to cancel the current alarm (if any).
	-> quiet	Reserved for future upgrade. This value is not currently used.
<b>Result</b>	0	No error.
	almErrMemory	Insufficient memory.
	almErrFull	Alarm table is full.

**Comments** This function sets an alarm for the specified application. An application can have only one alarm set at a time. If an alarm for this application has already been set, it is replaced with the new alarm. The alarmSeconds parameter specifies the time at which the alarm will be triggered. As soon as possible after this time, the alarm manager sends the [sysAppLaunchCmdAlarmTriggered](#) launch code to the specified application. If there is another alarm that should be set for this application, you can set it in response to this launch code. Following the [sysAppLaunchCmdAlarmTriggered](#) launch code, the alarm manager sends a [sysAppLaunchCmdDisplayAlarm](#) launch code. This is where your application should do things such as display a modal dialog

## Alarm Manager

### *Alarm Manager Functions*

---

indicating that the event has occurred. Read more about these launch codes in [Chapter 1, “Application Launch Codes.”](#)

If your application needs access to any particular value to respond to the alarm, pass a pointer to that value in the `ref` parameter. The system will pass this pointer back to the application in the launch codes’ parameter blocks.

**See Also** [AlmGetAlarm](#)

## AlmSetProcAlarm

**Purpose** Macro that sets or cancels a procedure alarm.

**Declared In** `AlarmMgr.h`

**Prototype** `AlmSetProcAlarm (procP, ref, alarmSeconds)`

**Parameters**

<code>-&gt; procP</code>	Pointer to a function that should be called when alarm is triggered. See <a href="#">AlmAlarmProcPtr</a> .
<code>-&gt; ref</code>	A caller-defined <code>UInt32</code> value. This value is passed with the launch code that notifies the application that the alarm has been triggered.
<code>-&gt; alarmSeconds</code>	A <code>UInt32</code> indicating the alarm date/time in seconds since 1/1/1904, or 0 to cancel the current alarm (if any).

**Result** Returns one of the following error codes:

0	No error.
<code>almErrMemory</code>	Insufficient memory.
<code>almErrFull</code>	Alarm table is full.

**Comments** This macro is similar to the [AlmSetAlarm](#) function, but it specifies a procedure to be called at the specified date and time rather than an application to be launched. With this macro, you can set alarms that are independent of any application. For example, a shared library

can set procedure alarms that call a procedure implemented in the library.

Procedure alarms also differ from regular system alarms in that if they trigger when the device is in sleep mode, the LCD does not light up. Thus, you can use procedure alarms to perform a scheduled task in a manner that is entirely hidden from the user.

---

**IMPORTANT:** Because the `procP` pointer is used to directly call the procedure, the pointer must remain valid from the time `AlmSetProcAlarm` is called to the time the alarm is triggered. If the procedure is in a shared library, you must keep the library open. If the procedure is in a separately loaded code resource, the resource must remain locked until the alarm is triggered. When you close a library or unlock a resource, you must remove any pending alarms. If you don't, the system will crash when the alarm is triggered.

---

**Compatibility** Implemented only if [3.2 New Feature Set](#) is present.

**See Also** [AlmGetProcAlarm](#)

## Application-Defined Functions

### AlmAlarmProcPtr

**Purpose** A procedure to be executed when an alarm is triggered.

**Declared In** `AlarmMgr.h`

**Prototype** `void (*AlmAlarmProcPtr) (UInt16 almProcCmd,  
SysAlarmTriggeredParamType *paramP)`

**Parameters** -> `almProcCmd` One of the `AlmProcCmdEnum` constants. These are commands that your function must handle. Possible values are:

## Alarm Manager

### *Application-Defined Functions*

---

almProcCmdTriggered

The alarm's date and time has passed and the alarm has been triggered. The function should perform its main task in response to this command.

almProcCmdReschedule

A system time change occurred, so the function must reschedule the alarm.

-> paramP

Pointer to a `SysAlarmTriggeredParamType` structure. See below.

**Result** Returns nothing.

**Comments** `AlmAlarmProcPtr` procedures are called when an alarm set by [AlmSetProcAlarm](#) is triggered. Your implementation should check the value of `almProcCmd` and respond accordingly.

The `paramP` parameter is a pointer to a `SysAlarmTriggeredParamType` structure. This structure is defined as:

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

`ref` and `alarmSeconds` are both values specified in [AlmSetProcAlarm](#) when the alarm is set. The `purgeAlarm` field specifies if the alarm will be removed from the alarm table when the function returns so that the [sysAppLaunchCmdDisplayAlarm](#) launch code is not triggered. This should be `true` for all procedure alarms; the alarm manager set it to `true` for you after your function returns.

If necessary, you can define new values for the `almProcCmd` parameter to call the procedure for something other than a triggered alarm or a system time change. The value you use must be greater than the constant `almProcCmdCustom` as defined in `AlarmMgr.h`.

**Compatibility** Implemented only if [3.2 New Feature Set](#) is present.

**See Also** [AlmGetProcAlarm](#)

## **Alarm Manager**

*Application-Defined Functions*

---

# Bitmaps

---

This chapter provides information about bitmaps by discussing these topics:

- [Bitmap Data Structures](#)
- [Bitmap Constants](#)
- [Bitmap Resources](#)
- [Bitmap Functions](#)

The header file `Bitmap.h` declares the API that this chapter describes. For more information on bitmaps, see the section “[Bitmaps](#)” on page 123 in the *Palm OS Programmer’s Companion*, vol. I.

## Bitmap Data Structures

### **BitmapCompressionType**

The `BitmapCompressionType` enum specifies possible bitmap compression types. These are the possible values for the `compressionType` field of [BitmapType](#). You can compress or uncompress a bitmap using a call to [BmpCompress](#).

```
typedef enum {
    BitmapCompressionTypeScanLine = 0,
    BitmapCompressionTypeRLE,
    BitmapCompressionTypePackBits,
    BitmapCompressionTypeEnd,
    BitmapCompressionTypeBest = 0x64,
    BitmapCompressionTypeNone = 0xFF
} BitmapCompressionType;
```

## Bitmaps

### Bitmap Data Structures

---

#### Value Descriptions

BitmapCompressionTypeScanLine	Use scan line compression. Scan line compression is compatible with Palm OS® 2.0 and higher.
BitmapCompressionTypeRLE	Use RLE compression. RLE compression is supported in Palm OS 3.5 and higher.
BitmapCompressionTypePackBits	Use PackBits compression. PackBits compression is supported in Palm OS 4.0 only.
BitmapCompressionTypeEnd	For internal use only.
BitmapCompressionTypeBest	For internal use only.
BitmapCompressionTypeNone	No compression is used.
	This value should only be used as an argument to <a href="#">BmpCompress</a> .

#### Compatibility

BitmapCompressionType is only defined if [3.5 New Feature Set](#) is present. Earlier releases do support compressed bitmaps, but in scan line format only.

## BitmapDirectInfoType

For direct color bitmaps—each pixel is represented by an RGB triplet rather than a palette index—the BitmapDirectInfoType structure follows the color table if one is present, or immediately follows the BitmapType if a color table is not present. For direct color bitmaps, only 16 bits per pixel is supported, 5 bits for red, 6 bits for green, and 5 bits for blue.

**WARNING!** This structure is documented so that you can directly access the internals of your own bitmap resources.

*Bitmaps created by Palm OS are not guaranteed to adhere to this structure*; you cannot cast a direct color bitmap data pointer from a bitmap created by the Palm OS to this structure and expect to be able to correctly access the structure's fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

---

```
typedef struct BitmapDirectInfoType {  
    UInt8 redBits;  
    UInt8 greenBits;  
    UInt8 blueBits;  
    UInt8 reserved;  
    RGBColorType transparentColor;  
} BitmapDirectInfoType;
```

### Field Descriptions

redBits	Number of bits used by the red component in each pixel.
greenBits	Number of bits used by the green component in each pixel.
blueBits	Number of bits used by the blue component in each pixel.
reserved	Must be zero. Reserved for future use.
transparentcolor	Contains the red, green, and blue components of the transparent color.

<b>Compatibility</b>	BitmapDirectInfoType is only defined if <a href="#">4.0 New Feature Set</a> is present.
----------------------	---

## BitmapFlagsType

The BitmapFlagsType bit field defines the flags field of [BitmapType](#). It specifies the bitmap's attributes.

## Bitmaps

### Bitmap Data Structures

---

**WARNING!** This structure is documented so that you can directly access the internals of your own bitmap resources.

*Bitmaps created by Palm OS are not guaranteed to adhere to this structure;* you cannot cast the `flags` field of a bitmap created by the Palm OS to this structure and expect to be able to correctly access the structure's fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

---

```
typedef struct BitmapFlagsType {  
    UInt16 compressed:1;  
    UInt16 hasColorTable:1;  
    UInt16 hasTransparency:1;  
    UInt16 indirect:1;  
    UInt16 forScreen:1;  
    UInt16 directColor:1;  
    UInt16 indirectColorTable:1;  
    UInt16 noDither:1;  
    UInt16 reserved:8;  
} BitmapFlagsType;
```

### Field Descriptions

compressed	If true, the bitmap is compressed and the <code>compressionType</code> field specifies the compression used. If false, the bitmap is uncompressed. The <a href="#">BmpCompress</a> function sets this field.
hasColorTable	If true, the bitmap has its own color table. If false, the bitmap uses the system color table. You specify whether the bitmap has its own color table when you create the bitmap.

hasTransparency	If <code>true</code> , the OS will not draw pixels that have a value equal to the <code>transparentIndex</code> . If <code>false</code> , the bitmap has no transparency value. You specify the transparent color when you create the bitmap using <code>Constructor</code> , or you can specify it programmatically with <a href="#"><u>BmpSetTransparentValue</u></a> . To obtain the value of this field, call <a href="#"><u>BmpGetTransparentValue</u></a> .
indirect	If <code>true</code> , the address to the bitmap's data is stored where the bitmap itself would normally be stored. The actual bitmap data is stored elsewhere. If <code>false</code> , the bitmap data is stored directly following the bitmap header or directly following the bitmap's color table if it has one. Never set this flag.  Note that this flag is supported for bitmaps created by Palm OS only; this flag is not used in user-created bitmap resources.
forScreen	If <code>true</code> , bitmap intended for the display (screen) window. Never set this flag.  Note that this flag is supported for bitmaps created by Palm OS only; this flag is not used in user-created bitmap resources.
directColor	If <code>true</code> , bitmap is a direct color (RGB) bitmap.

## Bitmaps

### Bitmap Data Structures

---

indirectColorTab If true, and if hasColorTable is true, a pointer to the bitmap's color table immediately follows the [BitmapType](#) structure. If false, and hasColorTable is true, the color table immediately follows the BitmapType structure. If hasColorTable is false, indirectColorTable is ignored. The indirect bit uses similar logic: if both the color table and the bitmap data are indirect, the color table pointer precedes the bitmap data pointer.

Note that this flag is supported for bitmaps created by Palm OS only; this flag is not used in user-created bitmap resources.

noDither If true, the blitter does not dither the bitmap. If false, the source bitmap is dithered if it has a bit depth greater than the destination bitmap.

reserved Reserved for future use.

### Compatibility

The hasTransparency, indirect, and forScreen flags are only defined if [3.5 New Feature Set](#) is present. The directColor flag is only defined if [4.0 New Feature Set](#) is present. The indirectColorTable and noDither flags are only defined if the [High-Density Display Feature Set](#) is present.

## BitmapPtr

The BitmapPtr type defines a pointer to a [BitmapType](#) structure.

```
typedef BitmapType *BitmapPtr;
```

## BitmapType

The BitmapType structure represents that which is common to all BitmapTypeVx structures ([BitmapTypeV0](#), [BitmapTypeV1](#), [BitmapTypeV2](#), and [BitmapTypeV3](#)). The BitmapType structures define both the bitmaps representing the window display and

bitmap resources ('Tbmp' and 'tAIB') that you create using Constructor or some other application and load into your program. Because BitmapType is merely a portion of the BitmapTypeVx structures, you should never do sizeof(BitmapType).

---

**WARNING!** This structure is documented so that you can directly access the internals of your own bitmap resources.

*Bitmaps created by Palm OS are not guaranteed to adhere to this structure;* you cannot cast a bitmap created by the Palm OS to this structure and expect to be able to correctly access the structure's fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

---

```
typedef struct BitmapType {  
    Int16 width;  
    Int16 height;  
    UInt16 rowBytes;  
    BitmapFlagsType flags;  
    UInt8 pixelSize;  
    UInt8 version;  
} BitmapType;  
  
typedef BitmapType* BitmapPtr;
```

### Field Descriptions

width	The width of the bitmap in pixels. You specify this value when you create the bitmap. Use <a href="#">BmpGetDimensions</a> to access this field.
height	The height of the bitmap in pixels. You specify this value when you create the bitmap. Use <a href="#">BmpGetDimensions</a> to access this field.
rowBytes	The number of bytes stored for each row of the bitmap where height is the number of rows. Use <a href="#">BmpGetDimensions</a> to obtain the contents of this field.

## Bitmaps

### Bitmap Data Structures

---

flags	The bitmap's attributes. See <a href="#">BitmapFlagsType</a> .
pixelSize	The pixel depth. Currently supported pixel depths are 1, 2, 4, and 8-bit. You specify this value when you create the bitmap. Use <a href="#">BmpGetBitDepth</a> to access the contents of this field.
version	The version of bitmap encoding used. See <a href="#">"Bitmap Constants"</a> on page 535. The value in this field determines the data structure to use when interpreting the fields following version: a value of BitmapVersionZero (0) corresponds to <a href="#">BitmapTypeV0</a> , BitmapVersionOne (1) corresponds to <a href="#">BitmapTypeV1</a> , BitmapVersionTwo (2) corresponds to <a href="#">BitmapTypeV2</a> , and BitmapVersionThree (3) corresponds to <a href="#">BitmapTypeV3</a> . Use <a href="#">BmpGetVersion</a> to obtain the contents of this field.

**Comments** Note the following about the `BitmapType` structures:

- None of these fields contains the actual bitmap data. Instead, the bitmap data is stored immediately following the `BitmapTypeVx` (which one depends on the value of the `version` field) header structure. If the bitmap has its own color table, the color table is stored in between the header and the data. If the bitmap has a pixel size of 16, and the bitmap is `BitmapTypeV2`, the `BitmapDirectInfoType` structure is stored between the header and the data. You can retrieve a bitmap's data by passing its `BitmapType` structure to `BmpGetBits`, and you can retrieve its color table with `BmpGetColortable`.
- Unlike most other user interface structures, the `BitmapType` does not store the bitmap's location on the screen. The `WindowType` or the `FormBitmapType` with which this bitmap is associated contains that information.
- A bitmap may be part of a bitmap family. A bitmap family is a group of bitmaps, each containing the same drawing but at

a different pixel depth (see “[Bitmaps](#)” on page 123 of the *Palm OS Programmer’s Companion*, vol. I). When requested to draw a bitmap family, the operating system chooses a member of the bitmap family based upon the bitmap density and pixel depth; see “[Bitmap Families](#)” on page 18 for the algorithm that the [High-Density Display Feature Set](#) uses to determine which one to choose.



---

## New [BitmapTypeV0](#)

Structure corresponding to the version 0 encoding of a bitmap. Version 0 encoding is supported in Palm OS 1.0 and later.

Generally you work with pointers to [BitmapType](#) structures; if the structure’s version is `BitmapVersionZero`, the structure is of type `BitmapTypeV0`.

---

**WARNING!** This structure is documented so that you can directly access the internals of your own bitmap resources.

*Bitmaps created by Palm OS are not guaranteed to adhere to this structure*; you cannot cast a bitmap created by the Palm OS to this structure and expect to be able to directly access the structure’s fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

---

```
typedef struct BitmapTypeV0 {  
    Int16 width;  
    Int16 height;  
    UInt16 rowBytes;  
    BitmapFlagsType flags;  
    UInt16 reserved[4];  
} BitmapTypeV0;  
  
typedef BitmapTypeV0 *BitmapPtrV0;
```

## Bitmaps

### Bitmap Data Structures

---

#### Field Descriptions

width	The width of the bitmap in pixels. You specify this value when you create the bitmap. Use <a href="#">BmpGetDimensions</a> to access this field.
height	The height of the bitmap in pixels. You specify this value when you create the bitmap. Use <a href="#">BmpGetDimensions</a> to access this field.
rowBytes	The number of bytes stored for each row of the bitmap where height is the number of rows. Use <a href="#">BmpGetDimensions</a> to access this field.
flags	The bitmap's attributes. See <a href="#">BitmapFlagsType</a> . Only the compressed flag is defined for BitmapTypeV0 structures.
reserved	Reserved. These values are set to zero. Note that in the BitmapTypeV0 structure, the pixelSize and version fields, defined in <a href="#">BitmapType</a> , do not exist. They coincide with the reserved array, however, and this array was initialized to zero when the bitmap was created. The operating system recognizes that a pixelSize of zero means that the bitmap's depth is 1.

#### Compatibility

BitmapTypeV0 is defined only if [High-Density Display Feature Set](#) is present.



## BitmapTypeV1

Structure corresponding to the version 1 encoding of a bitmap. Version 1 encoding is supported in Palm OS 3.0 and later.

Generally you work with pointers to [BitmapType](#) structures; if the structure's version is BitmapVersionOne, the structure is of type BitmapTypeV1.

**WARNING!** This structure is documented so that you can directly access the internals of your own bitmap resources.

*Bitmaps created by Palm OS are not guaranteed to adhere to this structure;* you cannot cast a bitmap created by the Palm OS to this structure and expect to be able to directly access the structure's fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

---

```
typedef struct BitmapTypeV1 {  
    Int16 width;  
    Int16 height;  
    UInt16 rowBytes;  
    BitmapFlagsType flags;  
    UInt8 pixelSize;  
    UInt8 version;  
    UInt16 nextDepthOffset;  
    UInt16 reserved[2];  
} BitmapTypeV1;  
  
typedef BitmapTypeV1* BitmapPtrV1;
```

## Field Descriptions

**width** The width of the bitmap in pixels. You specify this value when you create the bitmap. Use [BmpGetDimensions](#) to access this field.

**height** The height of the bitmap in pixels. You specify this value when you create the bitmap. Use [BmpGetDimensions](#) to access this field.

**rowBytes** The number of bytes stored for each row of the bitmap where height is the number of rows. Use [BmpGetDimensions](#) to access this field.

## Bitmaps

### Bitmap Data Structures

---

flags	The bitmap's attributes. See <a href="#">BitmapFlagsType</a> . Only the compressed and hasColorTable flags are defined for BitmapTypeV1 structures.
pixelSize	The pixel depth. Currently supported pixel depths are 1, 2, and 4-bit. You specify this value when you create the bitmap. Use <a href="#">BmpGetBitDepth</a> to obtain the contents of this field.
version	The version of bitmap encoding used. This field has a value of BitmapVersionOne (1) for BitmapTypeV1 structures. Use <a href="#">BmpGetVersion</a> to obtain the contents of this field.
nextDepthOffset	For bitmap families, this field specifies the start of the next bitmap in the family. The value it contains is the number of 4-byte words to the next BitmapType from the beginning of this one. If the bitmap is not part of a bitmap family or it is the last bitmap in the family, the nextDepthOffset is 0.
reserved	Reserved.

#### Compatibility

BitmapTypeV1 is defined only if [High-Density Display Feature Set](#) is present.



## BitmapTypeV2

---

Structure corresponding to the version 2 encoding of a bitmap. Version 2 encoding is supported in Palm OS 3.5 and later.

Generally you work with pointers to [BitmapType](#) structures; if the structure's version is BitmapVersionTwo (2), the structure is of type BitmapTypeV2.

**WARNING!** This structure is documented so that you can directly access the internals of bitmaps that you create. *Bitmaps created by Palm OS are not guaranteed to adhere to this structure*; you cannot cast a bitmap created by the Palm OS to this structure and expect to be able to directly access the structure's fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

---

```
typedef struct BitmapTypeV2 {  
    Int16 width;  
    Int16 height;  
    UInt16 rowBytes;  
    BitmapFlagsType flags;  
    UInt8 pixelSize;  
    UInt8 version;  
    UInt16 nextDepthOffset;  
    UInt8 transparentIndex;  
    UInt8 compressionType;  
    UInt16 reserved;  
} BitmapTypeV2;  
  
typedef BitmapTypeV2* BitmapPtrV2;
```

## Field Descriptions

width	The width of the bitmap in pixels. You specify this value when you create the bitmap. Use <a href="#">BmpGetDimensions</a> to access this field.
height	The height of the bitmap in pixels. You specify this value when you create the bitmap. Use <a href="#">BmpGetDimensions</a> to access this field.
rowBytes	The number of bytes stored for each row of the bitmap where height is the number of rows. Use <a href="#">BmpGetDimensions</a> to access this field.

## Bitmaps

### Bitmap Data Structures

---

flags	The bitmap's attributes. See <a href="#">BitmapFlagsType</a> . Only the compressed, hasColorTable, hasTransparency, indirect, forScreen, and directColor flags are defined for BitmapTypeV2 structures. Note that the indirect and forScreen flags are system-only flags that are not used in user-created bitmap resources.
pixelSize	The pixel depth. Currently supported pixel depths are 1, 2, 4, 8, and 16-bit. You specify this value when you create the bitmap. Use <a href="#">BmpGetBitDepth</a> to obtain the contents of this field.
version	The version of bitmap encoding used. This field has a value of BitmapVersionTwo (2) for BitmapTypeV2 structures. Use <a href="#">BmpGetVersion</a> to obtain the contents of this field.
nextDepthOffset	For bitmap families, this field specifies the start of the next bitmap in the family. The value it contains is the number of 4-byte words to the next BitmapType from the beginning of this one. If the bitmap is not part of a bitmap family or it is the last bitmap in the family, the nextDepthOffset is 0.
transparentIndex	The color index for the transparent color. Only used for version 2 bitmaps and only when the hasTransparency flag is set (see <a href="#">BitmapFlagsType</a> ). You specify this value when you create the bitmap using Constructor, or programmatically with <a href="#">BmpSetTransparentValue</a> . To obtain the value of this field, call <a href="#">BmpGetTransparentValue</a> .

compressionType The compression type used. Only used for version 2 bitmaps and only when the compressed flag is set (see [BitmapFlagsType](#)). See [BitmapCompressionType](#) for possible values. The [BmpCompress](#) function sets this field, and the [BmpGetCompressionType](#) function obtains its value.

reserved Reserved.

**Compatibility** BitmapTypeV2 is defined only if [High-Density Display Feature Set](#) is present.



## BitmapTypeV3

Structure corresponding to the version 3 encoding of a bitmap. Version 3 encoding is supported if the [High-Density Display Feature Set](#) is present.

Generally you work with pointers to [BitmapType](#) structures; if the structure's version is BitmapVersionThree (3), the structure is of type [BitmapTypeV3](#).

BmpCreate allocates and initializes a BitmapTypeV2 structure. To create a BitmapTypeV3 structure, use [BmpCreateBitmapV3](#) and supply the data pointer and optional color table pointer.

In earlier versions of the BitmapTypeVx structure, the size of compressed bitmap data is stored in a 16-bit field preceding the bitmap data. With the version 3 structure, the size is stored in a 32-bit field.

The BitmapTypeV3 structure has fields that identify how each pixel is stored (pixelFormat) and which color, if any, is "transparent" (transparentValue). Because of this, you don't use a [BitmapDirectInfoType](#) structure in conjunction with a BitmapTypeV3 structure.

## Bitmaps

### Bitmap Data Structures

---

**WARNING!** This structure is documented so that you can directly access the internals of bitmaps that you create. *Bitmaps created by Palm OS are not guaranteed to adhere to this structure*; you cannot cast a bitmap created by the Palm OS to this structure and expect to be able to directly access the structure's fields. Always use accessor functions to access the contents of user interface structures created by Palm OS.

---

```
typedef struct BitmapTypeV3 {  
    Int16 width;  
    Int16 height;  
    UInt16 rowBytes;  
    BitmapFlagsType flags;  
    UInt8 pixelSize;  
    UInt8 version;  
    UInt8 size;  
    UInt8 pixelFormat;  
    UInt8 unused;  
    UInt8 compressionType;  
    UInt16 density;  
    UInt32 transparentValue;  
    UInt32 nextBitmapOffset;  
} BitmapTypeV3;  
  
typedef BitmapTypeV3* BitmapPtrV3;
```

### Field Descriptions

width	The width of the bitmap in pixels. You specify this value when you create the bitmap. Use <a href="#">BmpGetDimensions</a> to access this field.
height	The height of the bitmap in pixels. You specify this value when you create the bitmap. Use <a href="#">BmpGetDimensions</a> to access this field.

rowBytes	The number of bytes stored for each row of the bitmap where height is the number of rows. Use <a href="#">BmpGetDimensions</a> to access this field.
flags	The bitmap's attributes. See <a href="#">BitmapFlagsType</a> . Note that the indirect, forScreen, and indirectColorTable flags are system-only fields that are not used in user-created bitmap resources.
pixelSize	The pixel depth. Currently supported pixel depths are 1, 2, 4, and 8-bit. You specify this value when you create the bitmap. Use <a href="#">BmpGetBitDepth</a> to obtain the value of this field.
version	The version of bitmap encoding used. This field has a value of BitmapVersionThree (3) for BitmapTypeV3 structures. The high bit of the version field is set if the bitmap data structure uses the native ARM format, with little-endian fields. Bitmap.h contains a bit mask, BitmapVersionMaskLE, that can be used to detect this. Use <a href="#">BmpGetVersion</a> to obtain the value of this field.
size	The size of this structure, in bytes. This field does not include the size of the color table or the size of the bitmap data. Use <a href="#">BmpGetSizes</a> to obtain the value of this field.
pixelFormat	An enumerated constant representing the format of the pixel data. See <a href="#">PixelFormatType</a> for the supported values.
unused	Not used.

## Bitmaps

### Bitmap Data Structures

---

compressionType	The compression type used; 0 if the bitmap is not compressed. Only used when the compressed flag is set (see <a href="#">BitmapFlagsType</a> ). See <a href="#">BitmapCompressionType</a> for possible values. The <a href="#">BmpCompress</a> function sets this field, and the <a href="#">BmpGetCompressionType</a> function obtains its value.
density	Value used by the blitter to determine how to stretch or shrink the bitmap data. For the screen bitmap, this field represents the screen density. For handhelds with low-density displays, this field is initialized to kDensityLow. For handhelds with double-density displays, this field is initialized to kDensityDouble. See <a href="#">DensityType</a> for the full set of values that this field can assume. Set this field with <a href="#">BmpSetDensity</a> , and obtain its value with <a href="#">BmpGetDensity</a> .
transparentValue	If this structure represents a bitmap with a bit depth of 8 or less, this field contains the bitmap's transparent index. If the bitmap has a bit depth of 16, the 16-bit transparent RGB color is stored in this field. You specify this value when you create the bitmap using Constructor, or programmatically with <a href="#">BmpSetTransparentValue</a> . To obtain the value of this field, call <a href="#">BmpGetTransparentValue</a> .
nextBitmapOffset	A 32-bit value that indicates the number of bytes to the next bitmap in the family. If the bitmap is not part of a bitmap family or it is the last bitmap in the family, the nextBitmapOffset is 0.
<b>Compatibility</b>	BitmapTypeV3 is defined only if <a href="#">High-Density Display Feature Set</a> is present.

## ColorTableType

The ColorTableType structure defines a color table. Bitmaps can have color tables attached to them; however, doing so is not recommended for performance reasons.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the ColorTableType structure. Never access its structure members directly, or your code may break in future versions. Use [BmpGetColortable](#) to access this structure. Use the information below for debugging purposes only.

---

```
typedef struct ColorTableType {  
    UInt16           numEntries;  
    // RGBColorType   entry[];  
} ColorTableType;
```

### Field Descriptions

**numEntries**      The number of entries in table. High bits (**numEntries** > 256) reserved.

The color table entries themselves are of type [RGBColorType](#), and there is one per **numEntries**. Use the macro [ColorTableEntries](#) to retrieve these entries.

Care should be taken not to confuse a full color table (which includes the count) with an array of RGB color values. Some routines operate on entire color tables; others operate on lists of color entries.

### Compatibility

ColorTableType is defined only if [3.5 New Feature Set](#) is present.



## DensityType

The density of the bitmap (see “[Display Density](#)” on page 75 of the *Palm OS Programmer’s Companion*, vol. I for a definition of display density). Density is only supported in [BitmapType](#) structures with

## Bitmaps

### Bitmap Data Structures

---

a version greater than 2; if a given `BitmapType` structure is version 2 or lower, it is assumed to contain low-density data.

The blitter uses the density field in the source and destination bitmaps to determine an appropriate scaling factor. When scaling down from a density of `kDensityDouble` to `kDensityLow`, the blitter must shrink the bitmap data. This will almost always result in a poorer-quality image when compared with a bitmap that was created with a density of `kDensityLow`.

The various `DensityType` values should not be interpreted as representing pixels per inch.

```
typedef enum {
    kDensityLow = 72,
    kDensityOneAndAHalf = 108,
    kDensityDouble = 144,
    kDensityTriple = 216,
    kDensityQuadruple = 288
} DensityType
```

### Value Descriptions

<code>kDensityLow</code>	Low (single) density. A low-density screen is 160x160 pixels.
<code>kDensityOneAndAHalf</code>	“One and a half” density. A one-and-a-half-density display is 240x240 pixels; this would most likely be used on a handheld with a 240x320 screen where the bottom portion is used as a “soft Graffiti” area.
<code>kDensityDouble</code>	Double density when compared with <code>kDensityLow</code> . A double-density screen is 320x320 pixels.
<code>kDensityTriple</code>	Triple density when compared with <code>kDensityLow</code> . A triple-density screen is 480x480 pixels.
<code>kDensityQuadruple</code>	Quadruple density when compared with <code>kDensityLow</code> . A quadruple-density screen is 640x640 pixels.

---

**IMPORTANT:** Not all densities listed in the `DensityType` enum are supported by a given version of the High-Density Display feature set. For Palm OS 5, only `kDensityLow` and `kDensityDouble` are supported.

---

**Compatibility** `DensityType` is defined only if [High-Density Display Feature Set](#) is present.



## PixelFormatType

Pixel formats defined for use with [BitmapTypeV3](#) structures.

```
typedef enum {
    pixelFormatIndexed,
    pixelFormat565
    pixelFormat565LE,
    pixelFormatIndexedLE
} PixelFormatType;
```

### Field Descriptions

`pixelFormatIndexed` Each pixel is represented by a palette index.

`pixelFormat565` Each pixel is represented by an RGB triplet stored in 16-bits: 5 red bits, 6 green bits, and 5 blue bits.

`pixelFormat565LE` Similar to `pixelFormat565`, except that the 16 bits of the RGB triplet are stored as little-endian. This pixel format is not supported in user-created bitmaps.

`pixelFormatIndexedLE` Similar to `pixelFormatIndexed`, except that the pixels within a byte are stored as little-endian. This pixel format is not supported in user-created bitmaps.

## Bitmaps

### Bitmap Data Structures

---

**Compatibility** PixelFormatType is defined only if [High-Density Display Feature Set](#) is present.

## RGBColorType

The RGBColorType structure defines a color. It is used as an entry in the color table. RGBColorTypes can also be created manually and passed to several user interface functions.

```
typedef struct RGBColorType {  
    UInt8 index;  
    UInt8 r;  
    UInt8 g;  
    UInt8 b;  
} RGBColorType;
```

### Field Descriptions

index	The index of this color in the color table. Not all functions that use RGBColorType use the index field. Direct bitmaps support no more than 256 colors. The number of possible RGB colors greatly exceeds this amount. For this reason, some drawing functions use a color look up table (CLUT). If the CLUT is used, the index field contains the index of an available color that is the closest match to the color specified by the r, g, and b fields.
r	Amount of red (0 to 255).
g	Amount of green (0 to 255).
b	Amount of blue (0 to 255).

**Compatibility** RGBColorType is defined only if [3.5 New Feature Set](#) is present.

## Bitmap Constants

Constant	Value	Description
BitmapVersionZero	0	Uses the version 0 encoding of a bitmap. Version 0 encoding is supported in Palm OS 1.0 and later.
BitmapVersionOne	1	Uses the version 1 encoding of a bitmap. Version 1 encoding is supported in Palm OS 3.0 and later.
		PalmRez automatically creates version 1 bitmaps unless you have specified a transparency index or a compressed type when creating the bitmap in Constructor.
BitmapVersionTwo	2	Uses the version 2 encoding of a bitmap. Palm OS 3.5 and later supports version 2 bitmaps. Version 2 bitmaps either use the transparency index or are compressed. If you programmatically create a bitmap using <a href="#">BmpCreate</a> , a version 2 bitmap is created.
BitmapVersionThree	3	Uses the version 3 encoding of a bitmap. Version 3 bitmaps are supported only if the <a href="#">High-Density Display Feature Set</a> is present.

## Bitmap Resources

You can create a bitmap resource and include it as part of your application's PRC file. Use the resource type 'Tbmp' for most images and the resource type 'tAIB' for application icons. Symbolically, these two resource types are `bitmapRsc` and `iconType`, respectively.

Note that if you are creating a bitmap or a bitmap family in Constructor, you create a 'tbmf' resource (or 'taif' resource for icons) and one or more 'PICT' images. The PalmRez post linker converts them into a single 'Tbmp' or 'tAIB' resource. Note that

## Bitmaps

### Bitmap Functions

---

the PalmRez post linker takes PICT images even on the Microsoft Windows operating system.

## Bitmap Functions

### BmpBitsSize

**Purpose** Return the size of the bitmap's data.

**Declared In** Bitmap.h

**Prototype** UInt16 BmpBitsSize (const BitmapType \*bitmapP)

**Parameters** -> bitmapP Pointer to the bitmap. (See [BitmapType](#).)

**Result** Returns the size in bytes of the bitmap's data, excluding the header and the color table.

**Comments** This function returns the bitmap's data size even if the bitmap's indirect flag is set. (See [BitmapFlagsType](#).)

If the bitmap is compressed, this function returns the compressed size of the bitmap.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [BmpSize](#), [BmpColortableSize](#), [BmpGetBits](#)

## BmpColortableSize

**Purpose** Return the size of the bitmap's color table.

**Declared In** Bitmap.h

**Prototype** `UInt16 BmpColortableSize  
(const BitmapType *bitmapP)`

**Parameters** `-> bitmapP` Pointer to the bitmap. (See [BitmapType](#).)

**Result** Returns the size in bytes of the bitmap's color table or 0 if the bitmap does not use its own color table.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [BmpBitsSize](#), [BmpSize](#), [BmpGetColortable](#)

## BmpCompress

**Purpose** Compress or uncompress a bitmap.

**Declared In** Bitmap.h

**Prototype** `Err BmpCompress (BitmapType *bitmapP,  
BitmapCompressionType compType)`

**Parameters** `-> bitmapP` Pointer to the bitmap to compress. (See [BitmapType](#).)

`-> compType` The type of compression to use. (See [BitmapCompressionType](#).) If set to `BitmapCompressionTypeNone` and `bitmapP` is compressed, this function uncompresses the bitmap.

**Result** Returns one of the following values:

`errNone` Success.

## Bitmaps

### Bitmap Functions

---

`sysErrParamErr` Either the `compType` parameter does not specify a compression type or the bitmap is already compressed, is in the storage heap, or represents the screen.

`sysErrNoFreeResource`

There is not enough memory available to complete the operation.

**Comments** This function performs the specified compression and resizes the bitmap's allocated memory. The bitmap must be in the dynamic heap.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [BmpGetCompressionType](#)

## BmpCreate

**Purpose** Create a bitmap.

**Declared In** Bitmap.h

**Prototype** `BitmapType *BmpCreate (Coord width, Coord height, UInt8 depth, ColorTableType *colortableP, UInt16 *error)`

**Parameters**

<code>-&gt; width</code>	The width of the bitmap in pixels. Must not be 0.
<code>-&gt; height</code>	The height of the bitmap in pixels. Must not be 0.
<code>-&gt; depth</code>	The pixel depth of the bitmap. Must be 1, 2, 4, 8, or 16. This value is used as the <code>pixelSize</code> field of <a href="#">BitmapType</a> .

-> colortableP A pointer to the color table associated with the bitmap, or NULL if the bitmap should not include a color table. If specified, the number of colors in the color table must match the depth parameter. (2 for 1-bit, 4 for 2-bit, 16 for 4-bit, and 256 for 8-bit). 16-bit bitmaps do not use a color table.

<- error Contains the error code if an error occurs.

**Result** Returns a pointer to the new bitmap structure (see [BitmapType](#)) or NULL if an error occurs. The parameter `error` contains one of the following:

`errNone` Success.

`sysErrParamErr` The width, height, depth, or `colorTableP` parameter is invalid. See the descriptions above for acceptable values.

`sysErrNoFreeResource`

There is not enough memory available to allocate the structure.

### Comments

This function creates an uncompressed, non-transparent `BitmapVersionTwo` bitmap with the width, height, and depth that you specify. To create a `BitmapVersionThree` bitmap use `BmpCreate` and pass the results to [BmpCreateBitmapV3](#).

If you pass a color table, the bitmap's `hasColorTable` flag is set. For performance reasons, attaching a custom color table to a bitmap is strongly discouraged. An alternative is to use the [WinPalette](#) command to change the color table as needed, draw the bitmap, and then undo your changes after you have finished displaying the bitmap.

`BmpCreate` allocates sufficient memory on the dynamic heap to hold the bitmap and initializes all of its pixels to white. To change the bitmap's contents, use the window drawing functions. First, you must use [WinCreateBitmapWindow](#) to create an off screen window wrapper around the bitmap, then draw to that window. For example:

## Bitmaps

### Bitmap Functions

---

```
BitmapType *bmpP;
WinHandle win;
Err error;
RectangleType onScreenRect;

bmpP = BmpCreate(10, 10, 8, NULL, &error);
if (bmpP) {
    win = WinCreateBitmapWindow(bmpP, &error);
    if (win) {
        WinSetDrawWindow(win);
        WinDrawLines(win, ...);
        /* etc */
        WinSetWindowBounds(win, onScreenRect);
    }
}
```

You cannot use this function to create a bitmap written directly to a database; that is, you must create the bitmap on the dynamic heap first, then write it to the storage heap.

It is not necessary to use BmpCreate to load a bitmap stored in a resource. Instead, you simply load the resource and lock its handle. The returned pointer is a pointer to a BitmapType. For example:

```
MemHandle resH =
    DmGetResource (bitmapRsc, rscID);
BitmapType *bitmap = MemHandleLock (resH);
```

Bitmaps 64 Kb and greater are now supported with Palm OS 4.0.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [BmpCreateBitmapV3](#), [BmpDelete](#)



**New**

## BmpCreateBitmapV3

**Purpose** Create a version 3 bitmap from an existing bitmap, an existing set of data bits, and, optionally, a color table.

**Declared In** Bitmap.h

**Prototype**

```
BitmapTypeV3 *BmpCreateBitmapV3
(const BitmapType *bitmapP, UInt16 density,
const void *bitsP,
const ColorTableType *colorTableP)
```

**Parameters**

- > bitmapP	Pointer to a valid bitmap from which the version 3 bitmap is to be created. See <a href="#">BitmapType</a> .
- > density	Density of the returned bitmap. If 0, the returned bitmap's density is set to the default value of kDensityLow.
- > bitsP	Pointer to the bitmap image data. Note that the bitmap data can be located in the storage heap, but then the bitmap should be treated as read-only. You must use DmWrite to write to the storage heap; blitting to it causes a system error.
- > colorTableP	Pointer to a color table, or NULL to use bitmapP's color table, if one exists.

**Result** Returns a version 3 bitmap, or NULL if the bitmap could not be created from the specified bitmap, bitmap data, and optional color table.

**Comments** You can use this function when the bitmap data is stored in the storage heap as bands of raster data. Rather than allocating several bitmap structures, one for each band, use this function to allocate a single bitmap, and have the structure point to each band successively. This is typically used with high-density bitmaps that cannot be stored entirely within 64k.

## Bitmaps

### Bitmap Functions

---

**WARNING!** Due to a limitation in the way that this function is implemented, `BitmapCreateBitmapV3` doesn't work with compressed bitmaps. Don't pass bitmaps to this function that have the compressed flag set.

---

The returned bitmap structure is allocated from the system heap. After your application is done with it, dispose of it by calling [BmpDelete](#).

**Compatibility** Implemented only if the [High-Density Display Feature Set](#) is present.

**See Also** [BmpCreate](#)

## BmpDelete

**Purpose** Delete a bitmap structure.

**Declared In** `Bitmap.h`

**Prototype** `Err BmpDelete (BitmapType *bitmapP)`

**Parameters** `-> bitmapP` Pointer to the structure of the bitmap to be deleted. (See [BitmapType](#).)

**Result** Returns `errNone` upon success, `sysErrParamErr` if the bitmap's `forScreen` flag is set or the bitmap resides in the storage heap. Returns one of the memory errors if the freeing pointer fails.

**Comments** Only delete bitmaps that have been created using [BmpCreate](#).

You cannot use this function on a bitmap located in a database. To delete a bitmap from a database, use the standard data manager calls.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

## BmpGetBits

<b>Purpose</b>	Retrieve the bitmap's data.
<b>Declared In</b>	Bitmap.h
<b>Prototype</b>	<code>void *BmpGetBits (BitmapType *bitmapP)</code>
<b>Parameters</b>	<code>-&gt; bitmapP</code> Pointer to the bitmap's structure. (See <a href="#">BitmapType</a> .)
<b>Result</b>	Returns a pointer to the bitmap's data.
<b>Comments</b>	This function returns the bitmap's data even if the bitmap's indirect flag is set. (See <a href="#">BitmapFlagsType</a> .)
<b>Compatibility</b>	Implemented only if <a href="#">3.5 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">BmpBitsSize</a>

## BmpGetBitDepth

<b>Purpose</b>	Retrieve the depth of a bitmap.
<b>Declared In</b>	Bitmap.h
<b>Prototype</b>	<code>UInt8 BmpGetBitDepth (const BitmapType* bitmapP)</code>
<b>Parameters</b>	<code>-&gt; bitmapP</code> Pointer to a bitmap. See <a href="#">BitmapType</a> .
<b>Result</b>	This function returns the bit depth of the bitmap, as represented by the <code>pixelSize</code> field in <a href="#">BitmapType</a> . For debug ROMs, this function reports an error and returns 0 if <code>bitmapP</code> is NULL.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present. To use this function in code intended to be run on earlier versions of Palm OS,

## Bitmaps

### Bitmap Functions

---

link with the PalmOSGlue library and call `BmpGlueGetBitDepth`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [BmpGetDimensions](#), [BmpGetNextBitmap](#), [BmpGetSizes](#)

## BmpGetColortable

**Purpose** Retrieve the bitmap's color table.

**Declared In** Bitmap.h

**Prototype** ColorTableType \*BmpGetColortable  
(BitmapType \*bitmapP)

**Parameters** -> bitmapP A pointer to the bitmap. See [BitmapType](#).

**Result** Returns a pointer to the color table or NULL if the bitmap uses the system color table.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [BmpColortableSize](#)



## BmpGetCompressionType

---

**Purpose** Get the compression type of a bitmap.

**Declared In** Bitmap.h

**Prototype** BitmapCompressionType BmpGetCompressionType  
(const BitmapType \*bitmapP)

**Parameters** -> bitmapP Pointer to a valid bitmap. See [BitmapType](#).

**Result** Returns the type of compression used by bitmapP. See [“BitmapCompressionType”](#) on page 513 for the values that can be

returned from this function. For debug ROMs, this function reports an error and returns BitmapCompressionTypeNone if bitmapP is NULL.

**Comments** If the bitmap is not compressed, this function returns BitmapCompressionTypeNone. If the bitmap version is 0 or 1 (corresponding to [BitmapTypeV0](#) and [BitmapTypeV1](#), respectively), it returns BitmapCompressionTypeScanLine.

**Compatibility** Implemented only if either the if [5.0 New Feature Set](#) or the [High-Density Display Feature Set](#) is present.

**See Also** [BmpCompress](#)



## New [BmpGetDensity](#)

---

**Purpose** Get the density of a bitmap.

**Declared In** Bitmap.h

**Prototype** UInt16 BmpGetDensity (const BitmapType \*bitmapP)

**Parameters** -> bitmapP Pointer to a valid bitmap. See [BitmapType](#).

**Result** Returns the density of bitmapP; see the [DensityType](#) enum for the defined set of density values. For debug ROMs, this function reports an error and returns 0 if bitmapP is NULL.

**Comments** Note that bitmaps with a version of 0, 1, or 2 (corresponding to [BitmapTypeV0](#), [BitmapTypeV1](#), and [BitmapTypeV2](#), respectively) are assumed to be low density (kDensityLow).

**Compatibility** Implemented only if the [High-Density Display Feature Set](#) is present.

**See Also** [BmpCreateBitmapV3](#), [BmpSetDensity](#)

## Bitmaps

### Bitmap Functions

---

## BmpGetDimensions

<b>Purpose</b>	Retrieve the width, height and number of data bytes per row of a bitmap.	
<b>Declared In</b>	Bitmap.h	
<b>Prototype</b>	<code>void BmpGetDimensions (const BitmapType *bitmapP, Coord *widthP, Coord *heightP, UInt16 *rowBytesP)</code>	
<b>Parameters</b>	<code>-&gt; bitmapP</code>	Pointer to the bitmap. See <a href="#">BitmapType</a> .
	<code>&lt;- widthP</code>	Pointer to bitmap's width in pixels. Use NULL if this information is not wanted.
	<code>&lt;- heightP</code>	Pointer to bitmap's height in pixels. Use NULL if this information is not wanted. Use NULL if this information is not wanted.
	<code>&lt;- rowBytesP</code>	Pointer to number of bytes per row of bitmap. Use NULL if this information is not wanted.
<b>Result</b>	This function returns the width in pixels of the bitmap in <code>widthP</code> , the height in pixels of the bitmap in <code>heightP</code> , and the number of bytes of data per row of the bitmap in <code>rowBytesP</code> . This function reports an error on debug ROMs if <code>bitmapP</code> is NULL.	
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call <code>BmpGlueGetDimensions</code> . For more information, see <a href="#">Chapter 75, "PalmOSGlue Library."</a>	
<b>See Also</b>	<a href="#">BmpGetBitDepth</a> , <a href="#">BmpGetNextBitmap</a> , <a href="#">BmpGetSizes</a>	

## BmpGetNextBitmap

**Purpose** Retrieve the next low-density bitmap in a bitmap family.

**Declared In** Bitmap.h

**Prototype** BitmapType \*BmpGetNextBitmap(BitmapType \*bitmapP)

**Parameters** -> bitmapP Pointer to a bitmap. See [BitmapType](#).

**Result** This function returns a pointer to the next low-density [BitmapType](#) in a bitmap family. It returns NULL if bitmapP is the last bitmap. For debug ROMs, this function reports an error and returns 0 if bitmapP is NULL.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call BmpGlueGetNextBitmap. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [BmpGetBitDepth](#), [BmpGetDimensions](#),  
[BmpGetNextBitmapAnyDensity](#), [BmpGetSizes](#)

**New**

## BmpGetNextBitmapAnyDensity

**Purpose** Get the next bitmap in the bitmap family, irrespective of density.

**Declared In** Bitmap.h

**Prototype** BitmapType \*BmpGetNextBitmapAnyDensity  
(BitmapType \*bitmapP)

**Parameters** -> bitmapP Pointer to a valid bitmap. See [BitmapType](#).

**Result** Returns the next bitmap in a bitmap family, or NULL if bitmapP is the last bitmap. For debug ROMs, this function reports an error and returns 0 if bitmapP is NULL.

**Comments** This function is an extended version of [BmpGetNextBitmap](#). For backward compatibility, BmpGetNextBitmap only returns low-density bitmaps. If the bitmap family contains high-density bitmaps, however, BmpGetNextBitmapAnyDensity skips over the dummy bitmap that separates the low and high-density bitmaps in the linked list and returns a high-density bitmap.

**Compatibility** Implemented only if the [High-Density Display Feature Set](#) is present.

**See Also** [BmpGetDensity](#), [BmpGetNextBitmap](#)

## BmpGetSizes

**Purpose** Retrieve the size of a bitmap and its header structure.

**Declared In** Bitmap.h

**Prototype** void BmpGetSizes (const BitmapType \*bitmapP,  
                  UInt32 \*dataSizeP, UInt32 \*headerSizeP)

**Parameters**

->bitmapP	Pointer to the bitmap. See <a href="#">BitmapType</a> .
<-dataSizeP	Pointer to size of bitmap data, not including structures. Use NULL if this information is not wanted.
<-headerSizeP	Pointer to size of bitmap's structures, not including data. Use NULL if this information is not wanted.

**Result** Returns the size of the bitmap and the size of the bitmap's structures. This function will report an error on debug ROMs if bitmapP is NULL.

**Comments** This function returns the size in bytes of the bitmap data in dataSizeP. The size does not include the data structures ([BitmapType](#), [BitmapDirectInfoType](#), or color table) that are associated with a bitmap. The size of the structures (in bytes) are returned in headerSizeP, which includes the size of the [BitmapType](#), [BitmapDirectInfoType](#) (if any), the color table (if any), and the size of the pointer for indirect bitmaps (described in [BitmapFlagsType](#)).

This function should be used when working with bitmaps that may be 64 Kb or greater. Do not use [BmpSize](#) or [BmpBitsSize](#) when working with bitmaps that may be greater than or equal to 64Kb.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [BmpGetBitDepth](#), [BmpGetDimensions](#), [BmpGetNextBitmap](#)

**New**

## BmpGetTransparentValue

**Purpose** Get a bitmap's transparent color.

**Declared In** Bitmap.h

**Prototype**

```
Boolean BmpGetTransparentValue  
(const BitmapType *bitmapP,  
 UInt32 *transparentValueP)
```

**Parameters**

-> bitmapP	Pointer to a valid bitmap. See <a href="#">BitmapType</a> .
<- transparentValueP	Pointer to a variable that receives the transparent color, either as a palette index or as a direct color value (an <a href="#">RGBColorType</a> ), depending on the bitmap's depth.

**Result** Returns true if bitmapP has a transparent color defined, false otherwise.

**Compatibility** Implemented only if either the if [5.0 New Feature Set](#) or the [High-Density Display Feature Set](#) is present.

**See Also** [BmpSetTransparentValue](#)



## New **BmpGetVersion**

**Purpose** Get the version of a bitmap.

**Declared In** Bitmap.h

**Prototype** UInt8 BmpGetVersion (const BitmapType \*bitmapP)

**Parameters** -> bitmapP Pointer to a valid bitmap. See [BitmapType](#).

**Result** Returns the version of bitmapP. See “[Bitmap Constants](#)” on page 535 for the defined bitmap version numbers. For debug ROMs, this function reports an error and returns 0 if bitmapP is NULL.

**Compatibility** Implemented only if the [High-Density Display Feature Set](#) is present.



## New **BmpSetDensity**

**Purpose** Set the density of a version 3 bitmap.

**Declared In** Bitmap.h

**Prototype** Err BmpSetDensity (BitmapType \*bitmapP,  
                  UInt16 density)

**Parameters** -> bitmapP Pointer to a valid version 3 bitmap. See [BitmapTypeV3](#).

## Bitmaps

### Bitmap Functions

---

-> density      The bitmap's density. This value should be one of the values defined by the [DensityType](#) enum.

**Result**      Returns errNone if the operation completed successfully, or SysErrParamErr either if bitmapP is NULL, if density is not supported by the blitter, or if \*bitmapP is not a version 3 bitmap.

**Comments**      To allocate a high-density bitmap, first call [BmpCreateBitmapV3](#). Then call BmpSetDensity to specify the bitmap's density.

**Compatibility**      Implemented only if the [High-Density Display Feature Set](#) is present.

**See Also**      [BmpGetDensity](#)



## New [BmpSetTransparentValue](#)

**Purpose**      Set a bitmap's transparent color.

**Declared In**      Bitmap.h

**Prototype**      void BmpSetTransparentValue (BitmapType \*bitmapP,  
                          UInt32 transparentValue)

**Parameters**      -> bitmapP      Pointer to a valid bitmap. See [BitmapType](#).

-> transparentValue      Transparent color. This should either be a palette index or a direct color value (an [RGBColorType](#)), depending on the bitmap's depth.

**Result**      Returns nothing.

**Comments**      If bitmapP points to a version 2 bitmap, BmpSetTransparentValue sets the [BitmapTypeV2](#) structure's

hasTransparency flag to true and initializes the structure's transparentIndex field according to transparentValue. For 16-bit bitmaps, this function sets the transparentColor field in the BitmapDirectInfoType auxiliary structure and sets the transparentIndex field to 0.

If bitmapP points to a version 3 bitmap, BmpSetTransparentValue sets the [BitmapTypeV3](#) structure's hasTransparency flag to true and sets the transparentValue field to the transparent color.

Regardless of the bitmap version, if this function is passed a transparentValue set to kTransparencyNone, this function sets the bitmap structure's hasTransparency flag to false and sets the transparent color field(s) to 0.

This function does nothing if transparentValue contains a value that is not valid for the depth of bitmapP.

**Compatibility**

Implemented only if either the if [5.0 New Feature Set](#) or the [High-Density Display Feature Set](#) is present.

**See Also**

[BmpGetTransparentValue](#)

## BmpSize

**Purpose** Return the size of the bitmap.

**Declared In** Bitmap.h

**Prototype** UInt16 BmpSize (const BitmapType \*bitmapP)

**Parameters** -> bitmapP A pointer to the bitmap. See [BitmapType](#).

**Result** Returns the size in bytes of the bitmap, including its header, color table (if any), and sizeof(BitmapDirectInfoType) if one exists.

**Comments** If the bitmap has its indirect flag set (see [BitmapFlagsType](#)), the bitmap data is not included in the size returned by this function.

## Bitmaps

### Bitmap Functions

---

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [BmpBitsSize](#), [BmpColortableSize](#)

## ColorTableEntries

**Purpose** Macro that returns the color table.

**Declared In** Bitmap.h

**Prototype** ColorTableEntries (ctP)

**Parameters** -> ctP A pointer to a [ColorTableType](#) structure.

**Result** Returns an array of [RGBColorType](#) structures, one for each entry in the color table.

**Comments** You can use this macro to retrieve the RGB values in use by a bitmap. For example:

```
BitmapType *bmpP;
RGBColorType *tableP =
    ColorTableEntries(BmpGetColorTable(bmpP));
```

If you want to retrieve the RGB values in use by the system color table, you can simply use the [WinPalette](#) function instead of this macro:

```
RGBColorType table[256];
Err e;

/* allocate space for table */
e = WinPalette(winPaletteGet, 0, 256, tableP);
```

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [BmpGetColortable](#)

# Character Attributes

---

This chapter provides reference material for character attributes functions defined in CharAttr.h.

## Character Attribute Functions

### ChrHorizEllipsis

**Purpose** Macro that returns the appropriate character code for the horizontal ellipsis.

**Declared In** Chars.h

**Prototype** ChrHorizEllipsis (chP)

**Parameters** <- chP      Pointer to a variable in which to return the horizontal ellipsis character code.

**Result** Returns nothing. Upon return, the variable pointed to by chP contains the horizontal ellipsis character.

**Comments** Version 3.1 of the Palm OS® uses different character codes for the horizontal ellipsis character and the numeric space character than earlier versions did. Use this macro to return the appropriate code for horizontal ellipsis regardless of which version of Palm OS your application is run on.

## ChrIsHardKey

<b>Purpose</b>	Macro that returns true if the character is one of the hard keys on the device.
<b>Declared In</b>	Chars.h
<b>Prototype</b>	ChrIsHardKey (ch)
<b>Parameters</b>	-> ch The character from the keyDownEvent.
<b>Result</b>	true if the character is one of the four built-in hard keys on the device, false otherwise.
<b>Compatibility</b>	This macro is obsolete and replaced by <a href="#">TxtCharIsHardKey</a> if the <a href="#">International Feature Set</a> is present.

## ChrNumericSpace

<b>Purpose</b>	Macro that returns the appropriate character code for the numeric space.
<b>Declared In</b>	Chars.h
<b>Prototype</b>	ChrNumericSpace (chP)
<b>Parameters</b>	<- chP Pointer to a variable in which to return the numeric space character code.
<b>Result</b>	Returns nothing. Upon return, the variable pointed to by chP contains the numeric space character.
<b>Comments</b>	Version 3.1 of the Palm OS uses different character codes for the horizontal ellipsis character and the numeric space character than earlier versions did. Use this macro to return the appropriate code for numeric space regardless of which version of Palm OS your application is run on.

## GetCharAttr

**Purpose** Return a pointer to the character attribute array. This array is used by the character classification and character conversion macros (such as `isalpha`).

**Declared In** CharAttr.h

**Prototype** UInt16\* GetCharAttr (void)

**Parameters** None

**Result** A pointer to the attributes array. This is an array of 256 `UInt16` values, one for each possible character code. See `CharAttr.h` for an explanation of the attributes.

**Compatibility** This function is **not** implemented if [International Feature Set](#) is present.

---

**NOTE:** This function is provided for backwards compatibility only. Use [Text Manager](#) functions instead on systems that support the text manager.

---

If [5.0 New Feature Set](#) is present, this function is implemented in PACE. However, it only supports the Latin table, regardless of the localization.

**See Also** [TxtCharAttr](#), [TxtCharXAttr](#)

## GetCharCaselessValue

<b>Purpose</b>	Return a pointer to an array that maps all characters to an assigned caseless and accentless value. Use this function for finding text.
<b>Declared In</b>	CharAttr.h
<b>Prototype</b>	UInt8* GetCharCaselessValue (void)
<b>Parameters</b>	None.
<b>Result</b>	Returns a pointer to the sort array, which is an array of 256 bytes.
<b>Comment</b>	<p>The GetCharCaselessValue conversion table converts each character into a numeric value that is caseless and sorted according to Microsoft Windows sorting rules:</p> <ul style="list-style-type: none"><li>• Punctuation characters have the lowest values,</li><li>• followed by numbers,</li><li>• followed by alpha characters.</li></ul> <p>All forms of each alpha character have equivalent values, so that e = E = e-grave = e-circumflex, etc.</p> <p>This conversion table is used by all the Palm OS sorting and comparison routines to yield caseless searches and caseless sorts in the almost same order as Windows-based programs, except that Palm OS routines produce the same sorting for all locales.</p>
<b>Compatibility</b>	This function is <b>not</b> implemented if <a href="#">International Feature Set</a> is present.

---

**NOTE:** This function is provided for backwards compatibility only. Use [Text Manager](#) functions instead on systems that support the text manager.

---

If [5.0 New Feature Set](#) is present, this function is implemented in PACE. However, it only supports the Latin table, regardless of the localization.

## GetCharSortValue

**Purpose** Return a pointer to an array that maps all characters to an assigned sorting value. Use this function for ordering (sorting) text.

**Declared In** CharAttr.h

**Prototype** UInt8\* GetCharSortValue (void)

**Parameters** None.

**Result** Returns a pointer to the attributes array. This is an array of 256 UInt8 values, one for each possible character code.

**Compatibility** This function is **not** implemented if [International Feature Set](#) is present.

---

**NOTE:** This function is provided for backwards compatibility only. Use [Text Manager](#) functions instead on systems that support the text manager.

---

If [5.0 New Feature Set](#) is present, this function is implemented in PACE. However, it only supports the Latin table, regardless of the localization.

## **Character Attributes**

*Character Attribute Functions*

---

# Data and Resource Manager

---

This chapter describes the data manager and the resource manager API declared in the header file `DataMgr.h`. It discusses the following topics:

- [Data Manager Data Structures](#)
- [Data Manager Constants](#)
- [Data Manager Functions](#)
- [Application-Defined Functions](#)

For more information on the data and resource managers, see the chapter “[Files and Databases](#)” in the *Palm OS Programmer’s Companion*, vol. I.

## Data Manager Data Structures

### DmOpenRef

The `DmOpenRef` type defines a pointer to an open database. The database pointer is created and returned by [`DmOpenDatabase`](#). It is used in any function that requires access to an open database.

```
typedef void *DmOpenRef
```

### DmResID

The `DmResID` type defines a resource identifier. You assign each resource an ID at creation time. Note that resource IDs greater than or equal to 10000 are reserved for system use.

## Data and Resource Manager

### *Data Manager Constants*

---

```
typedef UInt16 DmResID;
```

### DmResType

The DmResType type defines the type of a resource. The resource type is a four-character code such as 'Tbmp' for bitmap resources.

```
typedef UInt32 DmResType;
```

### SortRecordInfoType

The SortRecordInfoType structure specifies information that can be used to sort a record. The database sorting functions ([DmInsertionSort](#) and [DmQuickSort](#)) pass this structure to your comparison callback function (of type [DmComparF](#)), where you can use the information therein to help when comparing two records. To create this structure, you can call [DmRecordInfo](#), which returns these values for a given record.

```
typedef struct {
    UInt8 attributes;
    UInt8 uniqueID[3];
} SortRecordInfoType;
```

```
typedef SortRecordInfoType *SortRecordInfoPtr;
```

#### Field Descriptions

attributes      The record's attributes. See "[Record Attribute Constants](#)."

uniqueID      The unique identifier for the record.

## Data Manager Constants

### Category Constants

The following constants are used to specify information about categories:

<b>Constant</b>	<b>Value</b>	<b>Description</b>
dmAllCategories	0xFF	A mask used to represent all categories.
dmCategoryLength	16	The length of a category name. Currently, this is 16 bytes, which includes the null terminator.
dmRecAttrCategoryMask	0x0F	A mask used to retrieve the category information from the record's attributes field.
dmRecNumCategories	16	The number of categories allowed. Currently, this is 16, which includes the "Unfiled" category.
dmUnfiledCategory	0	A mask used to indicate the Unfiled category.

## Record Attribute Constants

The following constants specify a database record's attributes.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
dmMaxRecordIndex	0xFFFF	Indicates the highest record index allowed.
dmAllRecAttrs	0xF0	A mask used to specify all record attributes.
dmRecAttrBusy	0x20	Busy (the application has locked access to this record). A call to <a href="#">DmGetRecord</a> fails on a record that has this bit set, otherwise it sets this bit. Call <a href="#">DmReleaseRecord</a> to release the record and clear this bit. The <a href="#">DmSetRecordInfo</a> function cannot be used to alter the state of dmRecAttrBusy.
dmRecAttrDelete	0x80	Deleted
dmRecAttrDirty	0x40	Dirty (has been modified since last sync)
dmRecAttrSecret	0x10	Private
dmSysOnlyRecAttrs	0x20	A mask used to specify record attributes that only the system can change. (In other words, the busy attribute.)

## **Data and Resource Manager**

### *Data Manager Constants*

---

## Database Attribute Constants

The following constants define a database's attributes:

Constant	Description
dmAllHdrAttrs	A mask used to specify all header attributes.
dmHdrAttrAppInfoDirty	The application info block is dirty (has been modified since the last sync).
dmHdrAttrBackup	The database should be backed up to the desktop computer if no application-specific conduit is available.
dmHdrAttrBundle	The database is bundled with its application during a beam. That is, if the user chooses to beam the application from the Launcher, the Launcher beams this database along with the application's resource database and overlay database.
dmHdrAttrCopyPrevention	This attribute applies to Palm OS® 4.0 and higher. Note that overlay databases are automatically beamed with the application database on Palm OS 4.0 and higher. You do not need to set this bit in overlay databases.
dmHdrAttrHidden	Prevents the database from being copied by methods such as IR beaming.
	This database should be hidden from view. For example, this attribute is set to hide some applications in the launcher's main view. You can set it on record databases to have the launcher disregard the database's records when showing a count of records.
	This attribute applies to Palm OS version 3.2 and higher.

## Data and Resource Manager

### *Data Manager Constants*

---

Constant	Description
dmHdrAttrLaunchableData	This database is a data database but it can be “launched” from the launcher. For example, this attribute is set in Palm Query Applications (PQAs) launched by the Web Clipping Application Viewer application.
dmHdrAttrOpen	The database is open.
dmHdrAttrOKToInstallNewer	The backup conduit can install a newer version of this database with a different name if the current database is open. This mechanism is used to update the Graffiti® Shortcuts database, for example.
dmHdrAttrReadOnly	The database is a read-only database.
dmHdrAttrRecyclable	The database is recyclable. Recyclable databases are deleted when they are closed or upon a system reset.
	This attribute applies to Palm OS 4.0 and higher.
dmHdrAttrResDB	The database is a resource database.
dmHdrAttrResetAfterInstall	The device must be reset after this database is installed. That is, the HotSync® application forces a reset after installing this database.
dmHdrAttrStream	The database is a file stream.
dmSysOnlyHdrAttrs	A mask specifying the attributes that only the system can change (open and resource database).

## Error Codes

The following constants define error codes that are returned by the data manager and resource manager functions. Several functions return a failure value such as NULL or 0 instead of an error code. In many cases, you can call [DmGetLastErr](#) upon receiving this value and receive a more descriptive error code.

Also, note that on releases prior to Palm OS release 3.5, many data manager functions display a fatal error message using the [ErrFatalDisplayIf](#) macro if certain error conditions are true. Because the Palm OS ROMs are usually shipped with error checking set to partial, you receive the fatal error message. If a ROM is built with error checking set to none, the function returns one of the error codes listed here. (Note that Palm™ has never released a ROM with error checking set to none and has no plans to do so.)

<b>Constant</b>	<b>Description</b>
dmErrAlreadyExists	Another database with the same name already exists in RAM store.
dmErrAlreadyOpenForWrites	The database is already open with write access.
dmErrCantFind	The specified resource can't be found.
dmErrCantOpen	The database cannot be opened.
dmErrCorruptDatabase	The database is corrupted.
dmErrDatabaseOpen	The function cannot be performed on an open database, and the database is open.
dmErrDatabaseNotProtected	<a href="#"><u>DmDatabaseProtect</u></a> failed to protect the specified database.
dmErrIndexOutOfRange	The specified index is out of range.
dmErrInvalidDatabaseName	The name you've specified for the database is invalid.
dmErrInvalidParam	The function received an invalid parameter.
dmErrMemError	A memory error occurred.

## Data and Resource Manager

### *Data Manager Constants*

---

Constant	Description
dmErrNoOpenDatabase	The function is to search all open databases, but there are none.
dmErrNotRecordDB	You've attempted to perform a record function on a resource database.
dmErrNotResourceDB	You've attempted to perform a resource manager function on a record database.
dmErrNotValidRecord	The record handle is invalid.
dmErrOpenedByAnotherTask	You've attempted to open a database that another task already has open.
dmErrReadOnly	You've attempted to write to or modify a database that is in read-only mode.
dmErrRecordArchived	The function requires that the record not be archived, but it is.
dmErrRecordBusy	The function requires that the record not be busy, but it is.
dmErrRecordDeleted	The record has been deleted.
dmErrRecordInWrongCard	You've attempted to attach a record to a database when the record and database reside on different memory cards.

<b>Constant</b>	<b>Description</b>
dmErrResourceNotFound	The resource can't be found.
dmErrROMBased	You've attempted to delete or modify a ROM-based database.
dmErrSeekFailed	The operation of seeking the next record in the category failed.
dmErrUniqueIDNotFound	A record with the specified unique ID can't be found.
dmErrWriteOutOfBounds	A write operation exceeded the bounds of the record.
memErrCardNotPresent	The specified card can't be found.
memErrChunkLocked	The associated memory chunk is locked.
memErrInvalidParam	A memory error occurred.
memErrNotEnoughSpace	
memErrInvalidStoreHeader	The specified card has no storage RAM.
memErrRAMOnlyCard	
omErrBaseRequiresOverlay	An attempt was made to open a stripped resource database, but no associated overlay could be found.
omErrUnknownLocale	An attempt was made to open a resource database with overlays using an unknown locale.

---

## Data and Resource Manager

### *Data Manager Constants*

---

## Open Mode Constants

The following constants define the mode in which a database can be opened. You pass one or more of these as a parameter to [DmOpenDatabase](#), [DmOpenDatabaseByTypeCreator](#), or [DmOpenDBNoOverlay](#):

Constant	Description
dmModeReadWrite	Read-write access.
dmModeReadOnly	Read-only access.
dmModeWrite	Write-only access.
dmModeLeaveOpen	Leave database open even after application quits.
dmModeExclusive	Don't let anyone else open this database.
dmModeShowSecret	Show records marked private.

## Miscellaneous Constants

The following additional constants are used in conjunction with the Data Manager.

Constant	Value	Description
dmDBNameLength	32	Maximum length of a database name, including the null terminator. Database names must use only 7-bit ASCII characters (0x20 through 0x7E).

# Data Manager Functions

## DmArchiveRecord

**Purpose** Mark a record as archived by leaving the record's chunk intact and setting the delete bit for the next sync.

**Declared In** DataMgr.h

**Prototype** Err DmArchiveRecord (DmOpenRef dbP, UInt16 index)

**Parameters** -> dbP                    DmOpenRef to open database.  
              -> index                Which record to archive.

**Result** Returns errNone if no error, or one of the following if an error occurs:

- [dmErrReadOnly](#)
- [dmErrIndexOutOfRange](#)
- [dmErrRecordArchived](#)
- [dmErrRecordDeleted](#)
- [memErrInvalidParam](#)

Some releases may display a fatal error message instead of returning the error code.

**Comments** When a record is archived, the deleted bit is set but the chunk is not freed and the local ID is preserved. This way, the next time the user synchronizes with the desktop system, the desktop can save the record data on the PC before it permanently removes the record entry and data from the Palm Powered™ device.

Based on the assumption that a call to DmArchiveRecord indicates that you are finished with the record and aren't going to refer to it again, this function sets the chunk's lock count to zero.

**See Also** [DmRemoveRecord](#), [DmDetachRecord](#), [DmNewRecord](#),  
[DmDeleteRecord](#)

## Data and Resource Manager

### Data Manager Functions

---

## DmAttachRecord

**Purpose** Attach an existing chunk ID handle to a database as a record.

**Declared In** DataMgr.h

**Prototype** Err DmAttachRecord (DmOpenRef dbP, UInt16 \*atP,  
MemHandle newH, MemHandle \*oldHP)

**Parameters**

-> dbP	DmOpenRef to open database.
<-> atP	Pointer to the index where the new record should be placed. Specify the value dmMaxRecordIndex to add the record to the end of the database.
-> newH	Handle of the new record.
<-> oldHP	If non-NULL upon entry, indicates that the record at *atP should be replaced. Upon return, contains the handle to the replaced record.

**Result** Returns errNone if no error, or one of the following if an error occurs:

- [dmErrMemError](#)
- [memErrChunkLocked](#)
- [memErrInvalidParam](#)
- [memErrNotEnoughSpace](#)
- [dmErrReadOnly](#)
- [dmErrNotRecordDB](#)
- [dmErrRecordInWrongCard](#)
- [dmErrIndexOutOfRange](#)

Some releases may display a fatal error message instead of returning some of these error codes.

**Comments** Given the handle of an existing chunk, this routine makes that chunk a new record in a database and sets the dirty bit. The

parameter `atP` points to an index variable. If `oldHP` is NULL, the new record is inserted at index `*atP` and all record indices that follow are shifted down. If `*atP` is greater than the number of records currently in the database, the new record is appended to the end and its index is returned in `*atP`. If `oldHP` is not NULL, the new record replaces an existing record at index `*atP` and the handle of the old record is returned in `*oldHP` so that the application can free it or attach it to another database.

This function is useful for cutting and pasting between databases.

**See Also** [DmDetachRecord](#), [DmNewRecord](#), [DmNewHandle](#),  
[DmFindSortPosition](#)

## DmAttachResource

**Purpose** Attach an existing chunk ID to a resource database as a new resource.

**Declared In** DataMgr.h

**Prototype** Err DmAttachResource (DmOpenRef dbP,  
MemHandle newH, DmResType resType, DmResID resID)

**Parameters**

-> dbP	DmOpenRef to open database.
-> newH	Handle of new resource's data.
-> resType	Type of the new resource.
-> resID	ID of the new resource.

**Result** Returns `errNone` if no error, or one of the following if an error occurs:

- [dmErrMemError](#)
- [memErrChunkLocked](#)
- [memErrInvalidParam](#)
- [memErrNotEnoughSpace](#)
- [dmErrReadOnly](#)

## Data and Resource Manager

### Data Manager Functions

---

- [dmErrRecordInWrongCard](#)

Some releases may display a fatal error message instead of returning some of these error codes. All releases may display a fatal error message if the database is not a resource database.

**Comments** Given the handle of an existing chunk with resource data in it, this routine makes that chunk a new resource in a resource database. The new resource will have the given type and ID.

**See Also** [DmDetachResource](#), [DmRemoveResource](#), [DmNewHandle](#), [DmNewResource](#)

## DmCloseDatabase

**Purpose** Close a database.

**Declared In** DataMgr.h

**Prototype** Err DmCloseDatabase (DmOpenRef dbP)

**Parameters** -> dbP Database access pointer.

**Result** Returns errNone if no error, or [dmErrInvalidParam](#) if an error occurs. Some releases may display a fatal error message instead of returning the error code.

**Comments** This routine doesn't unlock any records that were left locked. Records and resources should not be left locked. If a record/resource is left locked, you should not use its reference because the record can disappear during a HotSync operation or if the database is deleted by the user. To prevent the database from being deleted, you can use [DmDatabaseProtect](#) before closing.

If there is an overlay associated with the database passed in, DmCloseDatabase closes the overlay as well.

If the database has the recyclable bit set (dmHdrAttrRecyclable), DmCloseDatabase calls [DmDeleteDatabase](#) to delete it.

**Compatibility** Starting with Palm OS 2.0, `DmCloseDatabase` updates the database's modification date.

- On Palm OS 2.0, the modification date is updated if the database was opened with write access.
- On Palm OS 3.0 and higher, the modification date is updated only if a change has been made and the database was opened with write access. Changes that trigger an update include adding, deleting, archiving, rearranging, or resizing records, setting a record's dirty bit in [DmReleaseRecord](#), rearranging or deleting categories, or updating the database header fields using [DmSetDatabaseInfo](#).

Under Palm OS 1.0, the modification date was never updated.

If you need to ensure that the modification date is updated the same way regardless of the operating system version, use [DmSetDatabaseInfo](#) to set the modification date explicitly.

**See Also** [DmOpenDatabase](#), [DmDeleteDatabase](#),  
[DmOpenDatabaseByTypeCreator](#)

## DmCreateDatabase

**Purpose** Create a new database on the specified card with the given name, creator, and type.

**Declared In** DataMgr.h

**Prototype** Err DmCreateDatabase (UInt16 cardNo,  
const Char \*nameP, UInt32 creator, UInt32 type,  
Boolean resDB)

**Parameters**

-> cardNo	The card number to create the database on.
-> nameP	Name of new database, up to 32 ASCII bytes long, including the null terminator (as specified by <code>dmDBNameLength</code> ). Database names must use only 7-bit ASCII characters (0x20 through 0x7E).
-> creator	Creator of the database.

## Data and Resource Manager

### *Data Manager Functions*

---

-> type                    Type of the database.  
-> resDB                If true, create a resource database.

**Result**      Returns errNone if no error, or one of the following if an error occurs:

- [dmErrInvalidDatabaseName](#)
- [dmErrAlreadyExists](#)
- [memErrCardNotPresent](#)
- [dmErrMemError](#)
- [memErrChunkLocked](#)
- [memErrInvalidParam](#)
- [memErrInvalidStoreHeader](#)
- [memErrNotEnoughSpace](#)
- [memErrRAMOnlyCard](#)

May display a fatal error message if the master database list cannot be found.

**Comments**     Call this routine to create a new database on a specific card. If another database with the same name already exists in RAM store, this routine returns a [dmErrAlreadyExists](#) error code. Once created, the database ID can be retrieved by calling [DmFindDatabase](#). The database can be opened using the database ID. To create a resource database instead of a record-based database, set the resDB Boolean to true.

After you create a database, it's recommended that you call [DmSetDatabaseInfo](#) to set the version number. Databases default to version 0 if the version isn't explicitly set.

**See Also**     [DmCreateDatabaseFromImage](#), [DmOpenDatabase](#),  
[DmDeleteDatabase](#)

## DmCreateDatabaseFromImage

<b>Purpose</b>	Create an entire database from a single resource that contains an image of the database.
<b>Declared In</b>	DataMgr.h
<b>Prototype</b>	Err DmCreateDatabaseFromImage (MemPtr bufferP)
<b>Parameters</b>	-> bufferP      Pointer to locked resource containing database image.
<b>Result</b>	Returns errNone if no error.
<b>Comments</b>	An image is the same as a desktop file representation of a prc or pdb file.  This function is intended for applications in the ROM to install default databases after a hard reset. RAM-based applications that want to install a default database should install a pdb file separately to save storage heap space.
<b>See Also</b>	<a href="#">DmCreateDatabase</a> , <a href="#">DmOpenDatabase</a>

## DmDatabaseInfo

<b>Purpose</b>	Retrieve information about a database.
<b>Declared In</b>	DataMgr.h
<b>Prototype</b>	Err DmDatabaseInfo (UInt16 cardNo, LocalID dbID, Char *nameP, UInt16 *attributesP, UInt16 *versionP, UInt32 *crDateP, UInt32 *modDateP, UInt32 *bckUpDateP, UInt32 *modNumP, LocalID *appInfoIDP, LocalID *sortInfoIDP, UInt32 *typeP, UInt32 *creatorP)
<b>Parameters</b>	-> cardNo      Number of the card the database resides on.

## Data and Resource Manager

### Data Manager Functions

---

-> dbID	Database ID of the database.
<- nameP	The database's name. Pass a pointer to 32-byte character array for this parameter, or NULL if you don't care about the name.
<- attributesP	The database's attribute flags. The section " <a href="#">Database Attribute Constants</a> " lists constants you can use to query the values returned in this parameter. Pass NULL for this parameter if you don't want to retrieve it.
<- versionP	The application-specific version number. The default version number is 0. Pass NULL for this parameter if you don't want to retrieve it.
<- crDateP	The date the database was created, expressed as the number of seconds since the first instant of Jan. 1, 1904. Pass NULL for this parameter if you don't want to retrieve it.
<- modDateP	The date the database was last modified, expressed as the number of seconds since the first instant of Jan. 1, 1904. Pass NULL for this parameter if you don't want to retrieve it.
<- bckUpDateP	The date the database was backed up, expressed as the number of seconds since the first instant of Jan. 1, 1904. Pass NULL for this parameter if you don't want to retrieve it.
<- modNumP	The modification number, which is incremented every time a record in the database is added, modified, or deleted. Pass NULL for this parameter if you don't want to retrieve it.
<- appInfoIDP	The local ID of the application info block, or NULL. The application info block is an optional field that the database may use to store application-specific information about the database. Pass NULL for this parameter if you don't want to retrieve it.

<- sortInfoIDP      The local ID of the database's sort table. This is an optional field in the database header. Pass NULL for this parameter if you don't want to retrieve it.

<- typeP      The database's type, specified when it is created. Pass NULL for this parameter if you don't want to retrieve it.

<- creatorP      The database's creator, specified when it is created. Pass NULL for this parameter if you don't want to retrieve it.

**Result**      Returns errNone if no error, or [dmErrInvalidParam](#) if an error occurs.

**Compatibility**      Updating of the modification date differs based on which version of the OS your application is running on.

- Under Palm OS 1.0, the modification date is never updated.
- Under Palm OS 2.0, the modification date is updated every time a database opened with write access is closed.
- Beginning with Palm OS 3.0, the modification date is updated only if a change has been made to the database opened with write access. (The update still occurs upon closing the database.) Changes that trigger an update include adding, deleting, archiving, rearranging, or resizing records, setting a record's dirty bit in [DmReleaseRecord](#), rearranging or deleting categories, or updating the database header fields using [DmSetDatabaseInfo](#).

If you need to ensure that the modification date is updated the same way regardless of the operating system version, use [DmSetDatabaseInfo](#) to set the modification date explicitly.

**See Also**      [DmSetDatabaseInfo](#), [DmDatabaseSize](#),  
[DmOpenDatabaseInfo](#), [DmFindDatabase](#),  
[DmGetNextDatabaseByTypeCreator](#),  
[TimSecondsToDate](#)

## Data and Resource Manager

### *Data Manager Functions*

---

## DmDatabaseProtect

**Purpose** Increment or decrement the database's protection count.

**Declared In** DataMgr.h

**Prototype** Err DmDatabaseProtect (UInt16 cardNo,  
LocalID dbID, Boolean protect)

**Parameters**

-> cardNo	Card number of database to protect/unprotect.
-> dbID	Local ID of database to protect/unprotect.
-> protect	If true, protect count will be incremented. If false, protect count will be decremented.

**Result** Returns errNone if no error, or one of the following if an error occurs:

- [memErrCardNotPresent](#)
- [dmErrROMBased](#)
- [dmErrCantFind](#)
- [memErrNotEnoughSpace](#)
- [dmErrDatabaseNotProtected](#)

**Comments** This routine can be used to prevent a database from being deleted (by passing true for the protect parameter). It increments the protect count if protect is true and decrements it if protect is false. All true calls should be balanced by false calls before the application terminates.

Use this function if you want to keep a particular record or resource in a database locked down but don't want to keep the database open. This information is kept in the dynamic heap, so all databases are "unprotected" at system reset.

If the database is a resource database that has an overlay associated with it for the current locale, the overlay is also protected or unprotected by this call.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present. Overlay support is only available if [3.5 New Feature Set](#) is present.

## DmDatabaseSize

**Purpose** Retrieve size information on a database.

**Declared In** DataMgr.h

**Prototype** Err DmDatabaseSize (UInt16 cardNo, LocalID dbID,  
                  UInt32 \*numRecordsP, UInt32 \*totalBytesP,  
                  UInt32 \*dataBytesP)

**Parameters**

-> cardNo	Card number the database resides on.
-> dbID	Database ID of the database.
<- numRecordsP	The total number of records in the database. Pass NULL for this parameter if you don't want to retrieve it.
<- totalBytesP	The total number of bytes used by the database including the overhead. Pass NULL for this parameter if you don't want to retrieve it.
<- dataBytesP	The total number of bytes used to store just each record's data, not including overhead. Pass NULL for this parameter if you don't want to retrieve it.

**Result** Returns errNone if no error, or [dmErrMemError](#) if an error occurs.

**See Also** [DmDatabaseInfo](#), [DmOpenDatabaseInfo](#), [DmFindDatabase](#),  
[DmGetNextDatabaseByTypeCreator](#)

## Data and Resource Manager

### Data Manager Functions

---

## DmDeleteCategory

**Purpose** Delete all records in a category. The category name is not changed.

**Declared In** DataMgr.h

**Prototype** Err DmDeleteCategory (DmOpenRef dbR,  
UInt16 categoryNum)

**Parameters**

-> dbR	Database access pointer.
-> categoryNum	Category of records to delete. Category masks such as dmAllCategories are invalid.

**Result** Returns errNone if no error, or one of the following if an error occurs:

- [dmErrReadOnly](#)
- [memErrInvalidParam](#)

Some releases may display a fatal error message instead of returning the error code.

**Comments** This function deletes all records in a category, but does not delete the category itself (note that it deletes the record data and header info, and doesn't just set the deleted bit). For each record in the category, DmDeleteCategory marks the delete bit in the database header for the record and disposes of the record's data chunk. The record entry in the database header remains, but its localChunkID is set to NULL.

If the category contains no records, this function does nothing and returns errNone to indicate success. The categoryNum parameter is assumed to represent a single category. If you pass a category mask to specify more than one category, this function interprets that value as a single category, finds no records to delete in that category, and returns errNone.

You can use the [DmRecordInfo](#) call to obtain a category index from a given record. For example:

```
DmOpenRef myDB;
UInt16 record, attr, category, total;

DmRecordInfo(myDB, record, &attr, NULL, NULL);
category = attr & dmRecAttrCategoryMask;
err = DmDeleteCategory(myDB, category);
```

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## DmDeleteDatabase

**Purpose** Delete a database and all its records.

**Declared In** DataMgr.h

**Prototype** Err DmDeleteDatabase (UInt16 cardNo, LocalID dbID)

**Parameters** -> cardNo Card number the database resides on.  
-> dbID Database ID.

**Result** Returns errNone if no error, or one of the following if an error occurs:

- [dmErrCantFind](#)
- [dmErrCantOpen](#)
- [memErrChunkLocked](#)
- [dmErrDatabaseOpen](#)
- [dmErrROMBased](#)
- [memErrInvalidParam](#)
- [memErrNotEnoughSpace](#)

**Comments** Call this routine to delete a database. This routine deletes the database, the application info block, the sort info block, and any other overhead information that is associated with this database. After deleting the database, this function enqueues a deferred [sysNotifyDBDeletedEvent](#) notification, which will be broadcast at the top of the event loop.

## Data and Resource Manager

### Data Manager Functions

---

If the database has an overlay associated with it, this function does **not** delete the overlay. You can delete the overlay with a separate call to `DmDeleteDatabase`.

This routine accepts a database ID as a parameter. To determine the database ID, call either `DmFindDatabase` or `DmGetDatabase` with a database index.

**Compatibility** The `sysNotifyDBDeletedEvent` notification is only broadcast if the [4.0 New Feature Set](#) is present.

**See Also** [DmDeleteRecord](#), [DmRemoveRecord](#), [DmRemoveResource](#), [DmCreateDatabase](#), [DmGetNextDatabaseByTypeCreator](#), [DmFindDatabase](#)

## DmDeleteRecord

**Purpose** Delete a record's chunk from a database but leave the record entry in the header and set the delete bit for the next sync.

**Declared In** DataMgr.h

**Prototype** Err DmDeleteRecord (DmOpenRef dbP, UInt16 index)

**Parameters** -> dbP                    DmOpenRef to open database.  
              -> index                  Which record to delete.

**Result** Returns errNone if no error, or one of the following if an error occurs:

- [dmErrReadOnly](#)
- [dmErrIndexOutOfRange](#)
- [dmErrRecordArchived](#)
- [dmErrRecordDeleted](#)
- [memErrInvalidParam](#)

Some releases may display a fatal error message instead of returning the error code.

**Comments** Marks the delete bit in the database header for the record and disposes of the record's data chunk. Does not remove the record entry from the database header, but simply sets the localChunkID of the record entry to NULL.

**See Also** [DmDetachRecord](#), [DmRemoveRecord](#), [DmArchiveRecord](#), [DmNewRecord](#)

## DmDetachRecord

**Purpose** Detach and orphan a record from a database but don't delete the record's chunk.

**Declared In** DataMgr.h

**Prototype** Err DmDetachRecord (DmOpenRef dbP, UInt16 index, MemHandle \*oldHP)

**Parameters**

-> dbP	DmOpenRef to open.
-> index	Index of the record to detach.
<-> oldHP	Pointer to return handle of the detached record.

**Result** Returns errNone if no error, or one of the following if an error occurs:

- [dmErrReadOnly](#)
- [dmErrIndexOutOfRange](#)
- [dmErrNotRecordDB](#)
- [memErrChunkLocked](#)
- [memErrInvalidParam](#)

Some releases may display a fatal error message instead of returning the error code.

**Comments** This routine detaches a record from a database by removing its entry from the database header and returns the handle of the record's data chunk in \*oldHP. Unlike [DmDeleteRecord](#), this

## Data and Resource Manager

### *Data Manager Functions*

---

routine removes its entry in the database header but it does not delete the actual record.

**See Also** [DmAttachRecord](#), [DmRemoveRecord](#), [DmArchiveRecord](#), [DmDeleteRecord](#)

## DmDetachResource

**Purpose** Detach a resource from a database and return the handle of the resource's data.

**Declared In** DataMgr.h

**Prototype** Err DmDetachResource (DmOpenRef dbP,  
                  UInt16 index, MemHandle \*oldHP)

**Parameters** -> dbP                      DmOpenRef to open database.  
                  -> index                  Index of resource to detach.  
                  <-> oldHP                 Pointer to return handle of the detached record.

**Result** Returns errNone if no error, or one of the following if an error occurs:

- [dmErrReadOnly](#)
- [dmErrIndexOutOfRange](#)
- [dmErrCorruptDatabase](#)
- [memErrChunkLocked](#)
- [memErrInvalidParam](#)

Some releases may display a fatal error message instead of returning the error code. All releases may display a fatal error message if the database is not a resource database.

**Comments** This routine detaches a resource from a database by removing its entry from the database header and returns the handle of the resource's data chunk in \*oldHP.

**See Also** [DmAttachResource](#), [DmRemoveResource](#)

## DmFindDatabase

**Purpose** Return the database ID of a database by card number and name.

**Declared In** DataMgr.h

**Prototype** LocalID DmFindDatabase (UInt16 cardNo,  
const Char \*nameP)

**Parameters** -> cardNo Number of card to search.  
-> nameP Name of the database to look for.

**Result** Returns the database ID. If the database can't be found, this function returns 0, and [DmGetLastErr](#) returns an error code indicating the reason for failure.

**See Also** [DmGetNextDatabaseByTypeCreator](#), [DmDatabaseInfo](#),  
[DmOpenDatabase](#)

## DmFindRecordByID

**Purpose** Return the index of the record with the given unique ID.

**Declared In** DataMgr.h

**Prototype** Err DmFindRecordByID (DmOpenRef dbP,  
UInt32 uniqueID, UInt16 \*indexP)

**Parameters** -> dbP Database access pointer.  
-> uniqueID Unique ID to search for.  
<- indexP Return index.

**Result** Returns 0 if found, otherwise [dmErrUniqueIDNotFound](#). May display a fatal error message if the unique ID is invalid.

**See Also** [DmQueryRecord](#), [DmGetRecord](#), [DmRecordInfo](#)

## Data and Resource Manager

### *Data Manager Functions*

---

## DmFindResource

<b>Purpose</b>	Search the given database for a resource by type and ID, or by pointer if it is non-NULL.								
<b>Declared In</b>	<code>DataMgr.h</code>								
<b>Prototype</b>	<code>UInt16 DmFindResource (DmOpenRef dbP, DmResType resType, DmResID resID, MemHandle resH)</code>								
<b>Parameters</b>	<table><tr><td><code>-&gt; dbP</code></td><td>Open resource database access pointer.</td></tr><tr><td><code>-&gt; resType</code></td><td>Type of resource to search for.</td></tr><tr><td><code>-&gt; resID</code></td><td>ID of resource to search for.</td></tr><tr><td><code>-&gt;resH</code></td><td>Pointer to locked resource, or NULL.</td></tr></table>	<code>-&gt; dbP</code>	Open resource database access pointer.	<code>-&gt; resType</code>	Type of resource to search for.	<code>-&gt; resID</code>	ID of resource to search for.	<code>-&gt;resH</code>	Pointer to locked resource, or NULL.
<code>-&gt; dbP</code>	Open resource database access pointer.								
<code>-&gt; resType</code>	Type of resource to search for.								
<code>-&gt; resID</code>	ID of resource to search for.								
<code>-&gt;resH</code>	Pointer to locked resource, or NULL.								
<b>Result</b>	Returns index of resource in resource database, or <code>0xFFFF</code> if not found.  May display a fatal error message if the database is not a resource database.								
<b>Comments</b>	Use this routine to find a resource in a particular resource database by type and ID or by pointer. It is particularly useful when you want to search only one database for a resource and that database is not the topmost one.								

---

**IMPORTANT:** This function searches for the resource only in the database you specify. If you pass a pointer to a base resource database, its overlay is **not** searched. To search both a base database and its overlay for a localized resource, use [DmGet1Resource](#) instead of this function.

---

If `resH` is `NULL`, the resource is searched for by type and ID.

If `resH` is not `NULL`, `resType` and `resID` are ignored and the index of the given locked resource is returned.

Once the index of a resource is determined, it can be locked down and accessed by calling [DmGetResourceIndex](#).

**See Also** [DmGetResource](#), [DmSearchResource](#), [DmResourceInfo](#),  
[DmGetResourceIndex](#), [DmFind ResourceType](#)

## DmFind ResourceType

**Purpose** Search the given database for a resource by type and type index.

**Declared In** DataMgr.h

**Prototype** UInt16 DmFind ResourceType (DmOpenRef dbP,  
DmResType resType, UInt16 typeIndex)

**Parameters** -> dbP Open resource database access pointer.  
-> resType Type of resource to search for.  
-> typeIndex Index of given resource type.

**Result** Index of resource in resource database, or 0xFFFF if not found.

May display a fatal error message if the database is not a resource database.

**Comments** Use this routine to retrieve all the resources of a given type in a resource database. By starting at typeIndex 0 and incrementing until an error is returned, the total number of resources of a given type and the index of each of these resources can be determined. Once the index of a resource is determined, it can be locked down and accessed by calling [DmGetResourceIndex](#).

## Data and Resource Manager

### *Data Manager Functions*

---

**IMPORTANT:** This function searches for resources only in the database you specify. If you pass a pointer to a base resource database, its overlay is **not** searched. To search both a base database and its overlay for a localized resource, use [DmGet1Resource](#) instead of this function.

---

**See Also** [DmGetResource](#), [DmSearchResource](#), [DmResourceInfo](#),  
[DmGetResourceIndex](#), [DmFindResource](#)

## DmFindSortPosition

**Purpose** Returns where in a sorted list of records a given record would be located. Useful to find where to insert a record with [DmAttachRecord](#). Uses a binary search.

**Declared In** DataMgr.h

**Prototype** `UInt16 DmFindSortPosition (DmOpenRef dbP,  
void *newRecord, SortRecordInfoPtr newRecordInfo,  
DmComparF *compar, Int16 other)`

**Parameters**

-> dbP	Database access pointer.
-> newRecord	Pointer to the new record.
-> newRecordInfo	Sort information about the new record. See <a href="#">SortRecordInfoType</a> .
-> compar	Pointer to comparison function. See <a href="#">DmComparF</a> .

-> other Any value the application wants to pass to the comparison function. This parameter is often used to indicate a sort direction (ascending or descending).

**Result** The position where the record should be inserted.

The position should be viewed as between the record returned and the record before it. Note that the return value may be one greater than the number of records.

**Comments** If newRecord has the same key as another record in the database, DmFindSortPosition assumes that newRecord should be inserted after that record. If there are several records with the same key, newRecord is inserted after all of them. For this reason, if you use DmFindSortPosition to search for the location of a record that you know is already in the database, you must subtract 1 from the result. (Be sure to check that the value is not 0.)

If there are deleted records in the database, DmFindSortPosition only works if those records are at the end of the database. DmFindSortPosition always assumes that a deleted record is greater than or equal to any other record.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

**See Also** [DmFindSortPositionV10](#)

## DmFindSortPositionV10

**Purpose** Return where a record should be. Useful to find where to insert a record with [DmAttachRecord](#). Uses a binary search.

**Declared In** DataMgr.h

**Prototype** `UInt16 DmFindSortPositionV10 (DmOpenRef dbP,  
void *newRecord, DmComparF *compar, Int16 other)`

**Parameters** -> dbP Database access pointer.

## Data and Resource Manager

### Data Manager Functions

---

-> newRecord	Pointer to the new record.
-> compar	Pointer to comparison function. See <a href="#">DmComparF</a> .
-> other	Any value the application wants to pass to the comparison function.

**Result** Returns the position where the record should be inserted. The position should be viewed as between the record returned and the record before it. Note that the return value may be one greater than the number of records.

**Comments** If there are deleted records in the database, DmFindSortPositionV10 only works if those records are at the end of the database. DmFindSortPositionV10 always assumes that a deleted record is greater than or equal to any other record.

**Compatibility** This function corresponds to the 1.0 version of DmFindSortPosition.

**See Also** [DmFindSortPosition](#), [DmQuickSort](#), [DmInsertionSort](#)

## DmGetAppInfoID

**Purpose** Return the local ID of the application info block.

**Declared In** DataMgr.h

**Prototype** LocalID DmGetAppInfoID (DmOpenRef dbP) .

**Parameters** -> dbP Database access pointer.

**Result** Returns local ID of the application info block. The application info block is an optional field that the database may use to store application-specific information about the database; if the database doesn't have an application info block, DmGetAppInfoID returns zero.

**See Also** [DmDatabaseInfo](#), [DmOpenDatabase](#)

## DmGetDatabase

<b>Purpose</b>	Return the database header ID of a database by index and card number.				
<b>Declared In</b>	DataMgr.h				
<b>Prototype</b>	<code>LocalID DmGetDatabase (UInt16 cardNo,                   UInt16 index)</code>				
<b>Parameters</b>	<table><tr><td>-&gt; cardNo</td><td>Card number of database.</td></tr><tr><td>-&gt; index</td><td>Index of database.</td></tr></table>	-> cardNo	Card number of database.	-> index	Index of database.
-> cardNo	Card number of database.				
-> index	Index of database.				
<b>Result</b>	Returns the database ID, or 0 if an invalid parameter is passed.				
<b>Comments</b>	<p>Call this routine to retrieve the database ID of a database by index. The index should range from 0 to <a href="#">DmNumDatabases</a>-1.</p> <p>This routine is useful for getting a directory of all databases on a card. The databases returned may reside in either the ROM or the RAM. The order in which databases are returned is not fixed; therefore, you should not rely on receiving a list of databases in a particular order.</p>				
<b>See Also</b>	<a href="#">DmOpenDatabase</a> , <a href="#">DmNumDatabases</a> , <a href="#">DmDatabaseInfo</a> , <a href="#">DmDatabaseSize</a>				

## DmGetDatabaseLockState

<b>Purpose</b>	Return information about the number of locked and busy records in a database.		
<b>Declared In</b>	DataMgr.h		
<b>Prototype</b>	<code>void DmGetDatabaseLockState (DmOpenRef dbR,                   UInt8 *highest, UInt32 *count, UInt32 *busy)</code>		
<b>Parameters</b>	<table><tr><td>-&gt; dbR</td><td>Database access pointer.</td></tr></table>	-> dbR	Database access pointer.
-> dbR	Database access pointer.		

## Data and Resource Manager

### *Data Manager Functions*

---

<- highest	The highest lock count found for all of the records in the database. If a database has two records, one has a lock count of 2 and one has a lock count of 1, the highest lock count is 2. Pass NULL for this parameter if you don't want to retrieve it.
<- count	The number of records that have the lock count that is returned in the highest parameter. Pass NULL for this parameter if you don't want to retrieve it.
<- busy	The number of records that have the busy bit set. Pass NULL for this parameter if you don't want to retrieve it.

**Result** No return value. Returns all information in the parameters you pass.

**Comments** This function is intended to be used for debugging purposes. You can use it to obtain information about how many records are busy and how much locking occurs.

**Compatibility** Implemented only if [3.2 New Feature Set](#) is present.

## DmGetLastErr

**Purpose** Return error code from last data manager call.

**Declared In** DataMgr.h

**Prototype** Err DmGetLastErr (void)

**Parameters** None.

**Result** Error code from last unsuccessful data manager call.

**Comments** Use this routine to determine why a data manager call failed. In particular, calls like [DmGetRecord](#) return 0 if unsuccessful, so

calling [DmGetLastErr](#) is the only way to determine why they failed.

Note that `DmGetLastErr` does not always reflect the error status of the last data manager call. Rather, it reflects the error status of data manager calls that don't return an error code. For some of those calls, the saved error code value is not set to 0 when the call is successful.

For example, if a call to [DmOpenDatabaseByTypeCreator](#) returns NULL for database reference (that is, it fails), `DmGetLastErr` returns something meaningful; otherwise, it returns the error value of some previous data manager call.

Only the following data manager functions currently affect the value returned by `DmGetLastErr`:

---

<a href="#">DmFindDatabase</a>	<a href="#">DmOpenDatabaseByTypeCreator</a>
<a href="#">DmOpenDatabase</a>	<a href="#">DmNewRecord</a>
<a href="#">DmQueryRecord</a>	<a href="#">DmGetRecord</a>
<a href="#">DmQueryNextInCategory</a>	<a href="#">DmPositionInCategory</a>
<a href="#">DmSeekRecordInCategory</a>	<a href="#">DmResizeRecord</a>
<a href="#">DmGetResource</a>	<a href="#">DmGet1Resource</a>
<a href="#">DmNewResource</a>	<a href="#">DmGetResourceIndex</a>
<a href="#">DmNewHandle</a>	<a href="#">DmOpenDBNoOverlay</a>
<a href="#">DmResizeResource</a>	

---

## Data and Resource Manager

### *Data Manager Functions*

---

## DmGetNextDatabaseByTypeCreator

**Purpose** Return a database header ID and card number given the type and/or creator. This routine searches all memory cards for a match.

**Declared In** DataMgr.h

**Prototype**

```
Err DmGetNextDatabaseByTypeCreator
(Boolean newSearch, DmSearchStatePtr stateInfoP,
UInt32 type, UInt32 creator,
Boolean onlyLatestVers, UInt16 *cardNoP,
LocalID *dbIDP)
```

**Parameters**

-> newSearch	true if starting a new search.
<-> stateInfoP	If newSearch is false, this must point to the same data used for the previous invocation.
-> type	Type of database to search for, pass 0 as a wildcard.
-> creator	Creator of database to search for, pass 0 as a wildcard.
-> onlyLatestVers	If true, only the latest version of a database with a given type and creator is returned.
<- cardNoP	On exit, the card number of the found database.
<- dbIDP	Database local ID of the found database.

**Result** Returns errNone if no error, or [dmErrCantFind](#) if no matches were found.

**Comments** You may need to call this function successively to discover all databases having a specified type/creator pair.  
To start the search, pass true for newSearch. Allocate a DmSearchStateType structure and pass it as the stateInfoP parameter. DmGetNextDatabaseByTypeCreator stores private information in stateInfoP and uses it if the search is continued.

To continue a search where the previous one left off, pass `false` for `newSearch` and pass the same `stateInfoP` that you used during the previous call to this function.

You can pass `NULL` as a wildcard operator for the `type` or `creator` parameter to conduct searches of wider scope. If the `type` parameter is `NULL`, this routine can be called successively to return all databases of the given creator. If the `creator` parameter is `NULL`, this routine can be called successively to return all databases of the given type. You can also pass `NULL` as the value for both of these parameters to return all available databases without regard to type or creator.

Because databases are scattered freely throughout memory space, they are not returned in any particular order—any database matching the specified type/creator criteria can be returned. Thus, if the value of the `onlyLatestVers` parameter is `false`, this function may return a database which is not the most recent version matching the specified type/creator pair. To obtain only the latest version of a database matching the search criteria, set the value of the `onlyLatestVers` parameter to `true`.

When determining which is the latest version of the database, RAM databases are considered newer than ROM databases that have the same version number. Because of this, you can replace any ROM-based application with your own version of it. Also, a RAM database on card 1 is considered newer than a RAM database on card 0 if the version numbers are identical.

---

**WARNING!** Don't create or delete a database while using `DmGetNextDatabaseByTypeCreator` to iterate through the existing databases. This could cause databases to be skipped, or it could result in a given database being returned more than once.

---

## Compatibility

In Palm OS version 3.1 and higher, if `onlyLatestVers` is `true`, you only receive one matching database for each type/creator pair. In version 3.0 and earlier, you could receive multiple matching databases if `onlyLatestVers` was `true`.

Note that the behavior is different only when you have specified a value for both `type` and `creator` and `onlyLatestVers` is `true`.

## Data and Resource Manager

### *Data Manager Functions*

---

For example, suppose your application maintains three databases that all have the same type, creator, and version number and you write this code to process them in some way:

```
DmSearchStateType state;
Boolean latestVer;
UInt16 card;
LocalID currentDB = 0;

theErr = DmGetNextDatabaseByTypeCreator(true,
    &state, myType, myCreator, latestVer, &card,
    &currentDB);
while (!theErr && currentDB) {
    /* do something with currentDB */
    /* now get the next DB */
    theErr = DmGetNextDatabaseByTypeCreator(
        false, &state, myType, myCreator,
        vlatestVer, &card, &currentDB);
}
```

If `latestVer` is `false`, then your code will work the same on all versions of Palm OS and will return all three databases whose type and creator match those specified. If `latestVer` is `true`, this code returns all three databases on Palm OS version 3.0 and earlier, but only returns one database on version 3.1 and higher. (Exactly which database it returns is unspecified.)

If you expect multiple databases to match your search criteria, make sure you call `DmGetNextDatabaseByTypeCreator` in one of the following ways to ensure that your code operates the same on all Palm OS versions:

- Set `onlyLatestVers` to `false` if you specify both a type and creator.
- Specify `NULL` for either the `type` or `creator` parameter (or both).

#### **See Also**

[DmGetDatabase](#), [DmFindDatabase](#), [DmDatabaseInfo](#),  
[DmOpenDatabaseByTypeCreator](#), [DmDatabaseSize](#)

## DmGetRecord

**Purpose** Return a handle to a record by index and mark the record busy.

**Declared In** DataMgr.h

**Prototype** MemHandle DmGetRecord (DmOpenRef dbP,  
                  UInt16 index)

**Parameters** -> dbP                      DmOpenRef to open database.  
                  -> index                      Which record to retrieve.

**Result** Returns a handle to record data. If another call to DmGetRecord for the same record is attempted before the record is released, NULL is returned and [DmGetLastErr](#) returns an error code indicating the reason for failure.

**Comments** Returns a handle to given record and sets the busy bit for the record.

If the record is ROM-based (pointer accessed), this routine makes a fake handle to it and stores this handle in the DmAccessType structure.

[DmReleaseRecord](#) should be called as soon as the caller finishes viewing or editing the record.

**See Also** [DmSearchRecord](#), [DmFindRecordByID](#), [DmRecordInfo](#),  
[DmReleaseRecord](#), [DmQueryRecord](#)

## Data and Resource Manager

### *Data Manager Functions*

---

## DmGetResource

**Purpose** Search all open resource databases and return a handle to a resource, given the resource type and ID.

**Declared In** DataMgr.h

**Prototype** MemHandle DmGetResource (DmResType type,  
DmResID resID)

**Parameters** -> type                 The resource type.  
                        ->resID                 The resource ID.

**Result** Handle to resource data. If the specified resource cannot be found, this function returns NULL and [DmGetLastErr](#) returns an error code indicating the reason for failure.

**Comments** Searches all open resource databases starting with the most recently opened one for a resource of the given type and ID. If found, the resource handle is returned. The application should call [DmReleaseResource](#) as soon as it finishes accessing the resource data. The resource handle is not locked by this function.

This function always returns the resource located in the overlay if any open overlay has a resource matching that type and ID. If there is no overlay version of the resource, this function returns the resource from the base database.

**See Also** [DmGet1Resource](#), [DmReleaseResource](#), [ResLoadConstant](#)

## DmGetResourceIndex

**Purpose** Return a handle to a resource by index.

**Declared In** DataMgr.h

**Prototype** MemHandle DmGetResourceIndex (DmOpenRef dbP,  
UInt16 index)

**Parameters** -> dbP Access pointer to open database.  
-> index Index of the resource whose handle you want.

**Result** Handle to resource data. If the specified index is out of range, this function returns NULL and [DmGetLastErr](#) returns an error code indicating the reason for failure.

May display a fatal error message if the database is not a resource database.

---

**IMPORTANT:** This function accesses the resource only in the database you specify. If you pass a pointer to a base resource database, its overlay is **not** accessed. Therefore, you should use care when using this function to access a potentially localized resource. You can use [DmSearchResource](#) to obtain a pointer to the overlay database if the resource is localized; however, it's more convenient to use [DmGetResource](#) or [DmGet1Resource](#).

---

**See Also** [DmFindResource](#), [DmFindResourceType](#), [DmSearchResource](#)

## Data and Resource Manager

### *Data Manager Functions*

---

## DmGet1Resource ✓

<b>Purpose</b>	Search the most recently opened resource database and return a handle to a resource given the resource type and ID.
<b>Declared In</b>	DataMgr.h
<b>Prototype</b>	MemHandle DmGet1Resource (DmResType type, DmResID resID)
<b>Parameters</b>	-> type                    The resource type. -> resID                  The resource ID.
<b>Result</b>	Handle to resource data. If unsuccessful, this function returns NULL and <a href="#">DmGetLastErr</a> returns an error code indicating the reason for failure.
<b>Comments</b>	Searches the most recently opened resource database for a resource of the given type and ID. If the database has an overlay associated with it, the overlay is searched first, and then the base database is searched if the overlay does not contain the resource. If found, the resource handle is returned. The application should call <a href="#">DmReleaseResource</a> as soon as it finishes accessing the resource data. The resource handle is not locked by this function.
<b>See Also</b>	<a href="#">DmGetResource</a> , <a href="#">DmReleaseResource</a> , <a href="#">ResLoadConstant</a>

## DmInsertionSort

<b>Purpose</b>	Sort records in a database.
<b>Declared In</b>	DataMgr.h
<b>Prototype</b>	Err DmInsertionSort (DmOpenRef dbR, DmComparF *compar, Int16 other)
<b>Parameters</b>	-> dbR                    Database access pointer.

-> compar	Comparison function. See <a href="#">DmComparF</a> .
-> other	Any value the application wants to pass to the comparison function. This parameter is often used to indicate a sort direction (ascending or descending).

<b>Result</b>	Returns errNone if no error, or one of the following if an error occurs:
	<ul style="list-style-type: none"><li>• <a href="#">dmErrReadOnly</a></li><li>• <a href="#">dmErrNotRecordDB</a></li></ul>
	Some releases may display a fatal error message instead of returning the error code.

<b>Comments</b>	Deleted records are placed last in any order. All others are sorted according to the passed comparison function. Only records which are out of order move. Moved records are moved to the end of the range of equal records. If a large number of records are being sorted, try to use the quick sort.
	The following insertion-sort algorithm is used: Starting with the second record, each record is compared to the preceding record. Each record not greater than the last is inserted into sorted position within those already sorted. A binary insertion is performed. A moved record is inserted after any other equal records.

<b>See Also</b>	<a href="#">DmQuickSort</a>
-----------------	-----------------------------

## DmMoveCategory

<b>Purpose</b>	Move all records in a category to another category.
----------------	---

<b>Declared In</b>	DataMgr.h
--------------------	-----------

<b>Prototype</b>	Err DmMoveCategory (DmOpenRef dbP, UInt16 toCategory, UInt16 fromCategory, Boolean dirty)
------------------	---

<b>Parameters</b>	-> dbP	DmOpenRef to open database.
-------------------	--------	-----------------------------

## Data and Resource Manager

### Data Manager Functions

---

->toCategory Category to which the records should be added.

-> fromCategory Category from which to remove records.

-> dirty If true, set the dirty bit.

**Result** Returns 0 if successful, or [dmErrReadOnly](#) if the database is in read-only mode. Some releases may display a fatal error message instead of returning the error code.

**Comments** If dirty is true, the moved records are marked as dirty.

The toCategory and fromCategory parameters hold category index values. You can learn which category a record is in with the [DmRecordInfo](#) call and use that value in this function. For example, the following code, ensures that the records rec1 and rec2 are in the same category:

```
DmOpenRef myDB;
UInt16 rec1, rec2;
UInt16 rec1Attr, rec2Attr;
UInt16 category1, category2;

DmRecordInfo (myDB, rec1, &rec1Attr, NULL,
    NULL) ;
category1 = rec1Attr & dmRecAttrCategoryMask;
DmRecordInfo(myDB, rec2, &rec2Attr, NULL,
    NULL) ;
category2 = rec2Attr & dmRecAttrCategoryMask;
if (category1 != category2)
    DmMoveCategory(myDB, category1, category2,
        true) ;
```

## DmMoveRecord

**Purpose** Move a record from one index to another.

**Declared In** DataMgr.h

**Prototype** Err DmMoveRecord (DmOpenRef dbP, UInt16 from,  
UInt16 to)

**Parameters**

-> dbP	DmOpenRef to open database.
-> from	Index of record to move.
-> to	Where to move the record.

**Result** Returns errNone if no error, or one of the following if an error occurs:

- [dmErrReadOnly](#)
- [dmErrIndexOutOfRange](#)
- [dmErrNotRecordDB](#)
- [dmErrMemError](#)
- [memErrInvalidParam](#)
- [memErrChunkLocked](#)

Some releases may display a fatal error message instead of returning the error code.

**Comments** Insert the record at the to index and move other records down. The to position should be viewed as an insertion position. This value may be one greater than the index of the last record in the database. In cases where to is greater than from, the new index of the record becomes to-1 after the move is complete.

## Data and Resource Manager

### Data Manager Functions

---

## DmNewHandle

**Purpose** Attempt to allocate a new chunk in the same data heap or card as the database header of the passed database access pointer. If there is not enough space in that data heap, try other heaps.

**Declared In** DataMgr.h

**Prototype** MemHandle DmNewHandle (DmOpenRef dbP, UInt32 size)

**Parameters**

-> dbP	DmOpenRef to open database.
-> size	Size of new handle.

**Result** Returns the chunkID of new chunk. If an error occurs, returns 0, and [DmGetLastErr](#) returns an error code indicating the reason for failure.

**Comments** Allocates a new handle of the given size. Ensures that the new handle is in the same memory card as the given database. This guarantees that you can attach the handle to the database as a record to obtain and save its LocalID in the appInfoID or sortInfoID fields of the header.  
  
The handle should be attached to a database as soon as possible. If it is not attached to a database and the application crashes, the memory used by the new handle is unavailable until the next soft reset.

## DmNewRecord

<b>Purpose</b>	Return a handle to a new record in the database and mark the record busy.							
<b>Declared In</b>	DataMgr.h							
<b>Prototype</b>	<pre>MemHandle DmNewRecord (DmOpenRef dbP, UInt16 *atP, UInt32 size)</pre>							
<b>Parameters</b>	<table><tr><td>-&gt; dbP</td><td>DmOpenRef to open database.</td></tr><tr><td>&lt;-&gt; atP</td><td>Pointer to index where new record should be placed. Specify the value dmMaxRecordIndex to add the record to the end of the database.</td></tr><tr><td>-&gt; size</td><td>Size of new record.</td></tr></table>		-> dbP	DmOpenRef to open database.	<-> atP	Pointer to index where new record should be placed. Specify the value dmMaxRecordIndex to add the record to the end of the database.	-> size	Size of new record.
-> dbP	DmOpenRef to open database.							
<-> atP	Pointer to index where new record should be placed. Specify the value dmMaxRecordIndex to add the record to the end of the database.							
-> size	Size of new record.							
<b>Result</b>	<p>Handle to record data. If an error occurs, this function returns 0 and <a href="#">DmGetLastErr</a> returns an error code indicating the reason for failure.</p> <p>Some releases may display a fatal error message if the database is opened in read-only mode or it is a resource database.</p>							
<b>Comments</b>	<p>Allocates a new record of the given size, and returns a handle to the record data. The parameter atP points to an index variable. The new record is inserted at index *atP and all record indices that follow are shifted down. If *atP is greater than the number of records currently in the database, the new record is appended to the end and its index is returned in *atP.</p> <p>Both the busy and dirty bits are set for the new record and a unique ID is automatically created.</p> <p><a href="#">DmReleaseRecord</a> should be called as soon as the caller finishes viewing or editing the record.</p>							
<b>See Also</b>	<a href="#">DmAttachRecord</a> , <a href="#">DmRemoveRecord</a> , <a href="#">DmDeleteRecord</a>							

## Data and Resource Manager

### *Data Manager Functions*

---

## DmNewResource

**Purpose** Allocate and add a new resource to a resource database.

**Declared In** DataMgr.h

**Prototype** MemHandle DmNewResource (DmOpenRef dbP,  
DmResType resType, DmResID resID, UInt32 size)

**Parameters**

-> dbP	DmOpenRef to open database.
-> resType	Type of the new resource.
-> resID	ID of the new resource.
-> size	Desired size of the new resource.

**Result** Returns a handle to the new resource. If an error occurs, this function returns NULL and [DmGetLastErr](#) returns an error code indicating the reason for failure.

May display a fatal error message if the database is not a resource database.

**Comments** Allocates a memory chunk for a new resource and adds it to the given resource database. The new resource has the given type and ID. If successful, the application should call [DmReleaseResource](#) as soon as it finishes initializing the resource.

**See Also** [DmAttachResource](#), [DmRemoveResource](#)

## DmNextOpenDatabase

<b>Purpose</b>	Return DmOpenRef to next open database for the current task.
<b>Declared In</b>	DataMgr.h
<b>Prototype</b>	DmOpenRef DmNextOpenDatabase (DmOpenRef currentP)
<b>Parameters</b>	-> currentP      Current database access pointer or NULL.
<b>Result</b>	DmOpenRef to next open database, or NULL if there are no more.
<b>Comments</b>	Call this routine successively to get the DmOpenRefs of all open databases. Pass NULL for currentP to get the first one. Applications don't usually call this function, but it is useful for system information.
<b>See Also</b>	<a href="#">DmOpenDatabaseInfo</a> , <a href="#">DmDatabaseInfo</a>

## DmNextOpenResDatabase

<b>Purpose</b>	Return access pointer to next open resource database in the search chain.
<b>Declared In</b>	DataMgr.h
<b>Prototype</b>	DmOpenRef DmNextOpenResDatabase (DmOpenRef dbP)
<b>Parameters</b>	-> dbP      Database reference, or 0 to start search from the top.
<b>Result</b>	Pointer to next open resource database.
<b>Comments</b>	Returns pointer to next open resource database. To get a pointer to the first one in the search chain, pass NULL for dbP. This is the database that is searched when <a href="#">DmGet1Resource</a> is called.

## Data and Resource Manager

### *Data Manager Functions*

---

If you use this function to access a resource database that might have an overlay associated with it, be careful how you use the result. The DmOpenRef returned by this function is a pointer to the overlay database, not the base database. If you subsequently pass this pointer to [DmFindResource](#), you'll receive a handle to the overlaid resource. If you're searching for a resource that is found only in the base, you won't find it. Instead, always use [DmGetResource](#) or [DmGet1Resource](#) to obtain a resource. Both of those functions search both the overlay databases and their associated base databases.

## DmNumDatabases

**Purpose** Determine how many databases reside on a memory card.

**Declared In** DataMgr.h

**Prototype** UInt16 DmNumDatabases (UInt16 cardNo)

**Parameters** -> cardNo              Number of the card to check.

**Result** The number of databases found.

**Comments** This routine is helpful for getting a directory of all databases on a card. The routine [DmGetDatabase](#) accepts an index from 0 to [DmNumDatabases](#) -1 and returns a database ID by index.

**See Also** [DmGetDatabase](#)

## DmNumRecords

<b>Purpose</b>	Return the number of records in a database.
<b>Declared In</b>	DataMgr.h
<b>Prototype</b>	UInt16 DmNumRecords (DmOpenRef dbP)
<b>Parameters</b>	-> dbP                      DmOpenRef to open database.
<b>Result</b>	The number of records in a database.
<b>Comments</b>	Records that have that have the deleted bit set (that is, records that will be deleted during the next synchronization because the user has marked them deleted) are included in the count. If you want to exclude these records from your count, use <a href="#">DmNumRecordsInCategory</a> and pass dmAllCategories as the category.
<b>See Also</b>	<a href="#">DmNumRecordsInCategory</a> , <a href="#">DmRecordInfo</a> , <a href="#">DmSetRecordInfo</a>

## DmNumRecordsInCategory

<b>Purpose</b>	Return the number of records of a specified category in a database.
<b>Declared In</b>	DataMgr.h
<b>Prototype</b>	UInt16 DmNumRecordsInCategory (DmOpenRef dbP, UInt16 category)
<b>Parameters</b>	-> dbP                      DmOpenRef to open database. -> category                 Category index.
<b>Result</b>	The number of records in the category.

## Data and Resource Manager

### Data Manager Functions

---

- Comments** Because this function must examine all records in the database, it can be slow to return, especially when called on a large database. Records that have the deleted bit set are not counted, and if the user has specified to hide or mask private records, private records are not counted either.
- You can use the [DmRecordInfo](#) call to obtain a category index from a given record. For example:

```
DmOpenRef myDB;  
UInt16 record, attr, category, total;  
  
DmRecordInfo(myDB, record, &attr, NULL, NULL);  
category = attr & dmRecAttrCategoryMask;  
total = DmNumRecordsInCategory(myDB, category);
```

- See Also** [DmNumRecords](#), [DmQueryNextInCategory](#),  
[DmPositionInCategory](#), [DmSeekRecordInCategory](#),  
[DmMoveCategory](#)

## DmNumResources

- Purpose** Return the total number of resources in a given resource database.
- Declared In** DataMgr.h
- Prototype** UInt16 DmNumResources (DmOpenRef dbP)
- Parameters** -> dbP                    DmOpenRef to open database.
- Result** The total number of resources in the given database.  
May display a fatal error message if the database is not a resource database.
- Comments** DmNumResources only counts the resources in the database indicated by the DmOpenRef parameter. If the database is a resource database that has an overlay associated with it, this function only returns the number of resources in the base database, not in the overlay.

## DmOpenDatabase

<b>Purpose</b>	Open a database and return a reference to it. If the database is a resource database, also open its overlay for the current locale.						
<b>Declared In</b>	<code>DataMgr.h</code>						
<b>Prototype</b>	<code>DmOpenRef DmOpenDatabase (UInt16 cardNo, LocalID dbID, UInt16 mode)</code>						
<b>Parameters</b>	<table><tr><td>-&gt; cardNo</td><td>Card number database resides on.</td></tr><tr><td>-&gt; dbID</td><td>The database ID of the database.</td></tr><tr><td>-&gt; mode</td><td>Which mode to open database in (see "<a href="#">Open Mode Constants</a>").</td></tr></table>	-> cardNo	Card number database resides on.	-> dbID	The database ID of the database.	-> mode	Which mode to open database in (see " <a href="#">Open Mode Constants</a> ").
-> cardNo	Card number database resides on.						
-> dbID	The database ID of the database.						
-> mode	Which mode to open database in (see " <a href="#">Open Mode Constants</a> ").						
<b>Result</b>	Returns <code>DmOpenRef</code> to open database. May display a fatal error message if the database parameter is NULL. On all other errors, this function returns 0 and <a href="#"><code>DmGetLastErr</code></a> returns an error code indicating the reason for failure.						
<b>Comments</b>	<p>Call this routine to open a database for reading or writing.</p> <p>This routine returns a <code>DmOpenRef</code> which must be used to access particular records in a database. If unsuccessful, 0 is returned and the cause of the error can be determined by calling <a href="#"><code>DmGetLastErr</code></a>.</p> <p>When you use this routine to open a resource database in read-only mode, it also opens the overlay associated with this database for the current locale, if it exists. (The function <a href="#"><code>OmGetCurrentLocale</code></a> returns the current locale.) Overlays are resource databases typically used to localize applications, shared libraries, and panels. They have the same creator as the base database, a type of 'ovly' (symbolically named <code>omOverlayDBType</code>), and contain resources with the same IDs and types as the resources in the base database. When you request a resource from the database using <a href="#"><code>DmGetResource</code></a> or <a href="#"><code>DmGet1Resource</code></a>, the overlay is searched first. If the overlay contains a resource for the given ID, it is returned. If not, the resource from the base database is returned.</p>						

## Data and Resource Manager

### Data Manager Functions

---

The `DmOpenRef` returned by this function is the pointer to the base database, not to the overlay database, so care should be taken when passing this pointer to functions such as [DmFindResource](#) because this circumvents the overlay.

It's possible to create a "stripped" base resource database, one that does not contain any user interface resources. `DmOpenDatabase` only opens a stripped database if its corresponding overlay exists. If the overlay does not exist or if the overlay doesn't match the resource database, `DmOpenDatabase` returns NULL and [DmGetLastErr](#) returns the error code [omErrBaseRequiresOverlay](#).

If you open a resource database in a writable mode, the associated overlay is not opened. If you make changes to the resource database, the overlay database is invalidated if those changes affect any resources that are also in the overlay. This means that on future occasions where you open the resource database in read-only mode, the overlay will not be opened because Palm OS considers it to be invalid.

---

**TIP:** If you want to prevent your resource database from being overlaid, include an '`xprf`' resource (symbolically named `sysResTExtPrefs`) in the database with the ID 0 (`sysResIDExtPrefs`) and set its `disableOverlays` flag. This resource is defined in `UIResources.r`.

---

#### Compatibility

Overlay support is only available if [3.5 New Feature Set](#) is present. On earlier releases, this function opens resource databases without looking for an associated overlay.

If [4.0 New Feature Set](#) is present, when `DmOpenDatabase` attempts to open a stripped resource database and cannot find an overlay for it, it searches for an overlay matching the default locale if the system locale is different from the default locale.

#### See Also

[DmCloseDatabase](#), [DmCreateDatabase](#), [DmFindDatabase](#), [DmOpenDatabaseByTypeCreator](#), [DmDeleteDatabase](#), [DmOpenDBNoOverlay](#)

## DmOpenDatabaseByTypeCreator

<b>Purpose</b>	Open the most recent revision of a database with the given type and creator. If the database is a resource database, also open its overlay for the current locale.						
<b>Declared In</b>	<code>DataMgr.h</code>						
<b>Prototype</b>	<code>DmOpenRef DmOpenDatabaseByTypeCreator (UInt32 type, UInt32 creator, UInt16 mode)</code>						
<b>Parameters</b>	<table><tr><td>-&gt; type</td><td>Type of database.</td></tr><tr><td>-&gt; creator</td><td>Creator of database.</td></tr><tr><td>-&gt; mode</td><td>Which mode to open database in (see “<a href="#">Open Mode Constants</a>”).</td></tr></table>	-> type	Type of database.	-> creator	Creator of database.	-> mode	Which mode to open database in (see “ <a href="#">Open Mode Constants</a> ”).
-> type	Type of database.						
-> creator	Creator of database.						
-> mode	Which mode to open database in (see “ <a href="#">Open Mode Constants</a> ”).						
<b>Result</b>	<code>DmOpenRef</code> to open database. If the database couldn't be found, this function returns 0 and <a href="#">DmGetLastErr</a> returns an error code indicating the reason for failure.						
<b>Comments</b>	If you use this routine to open a resource database in read-only mode, it also opens the overlay associated with this database for the current locale. See <a href="#">DmOpenDatabase</a> for more information on overlays and resource databases.						
<b>Compatibility</b>	Overlay support is only available if <a href="#">3.5 New Feature Set</a> is present. On earlier releases, this function opens resource databases without looking for an associated overlay.						
<b>See Also</b>	<a href="#">DmCreateDatabase</a> , <a href="#">DmOpenDatabase</a> , <a href="#">DmOpenDatabaseInfo</a> , <a href="#">DmCloseDatabase</a> , <a href="#">DmOpenDBNoOverlay</a>						

## Data and Resource Manager

### Data Manager Functions

---

## DmOpenDatabaseInfo

**Purpose** Retrieve information about an open database.

**Declared In** DataMgr.h

**Prototype** Err DmOpenDatabaseInfo (DmOpenRef dBp,  
LocalID \*dbIDP, UInt16 \*openCountP, UInt16 \*modeP,  
UInt16 \*cardNoP, Boolean \*resDBP)

<b>Parameters</b>	-> dBp	DmOpenRef to open database.
	<- dbIDP	The ID of the database. Pass NULL for this parameter if you don't want to retrieve this information.
	<- openCountP	The number of applications that have this database open. Pass NULL for this parameter if you don't want to retrieve this information.
	<- modeP	The mode used to open the database (see “ <a href="#">Open Mode Constants</a> ”). Pass NULL for this parameter if you don't want to retrieve this information.
	<- cardNoP	The number of the card on which this database resides. Pass NULL for this parameter if you don't want to retrieve this information.
	<- resDBP	If true upon return, the database is a resource database, false otherwise. Pass NULL for this parameter if you don't want to retrieve this information.

**Result** Returns errNone if no error.

**See Also** [DmDatabaseInfo](#)

## DmOpenDBNoOverlay

<b>Purpose</b>	Open a database and return a reference to it.						
<b>Declared In</b>	<code>DataMgr.h</code>						
<b>Prototype</b>	<code>DmOpenRef DmOpenDBNoOverlay (UInt16 cardNo, LocalID dbID, UInt16 mode)</code>						
<b>Parameters</b>	<table><tr><td>-&gt; cardNo</td><td>Card number database resides on.</td></tr><tr><td>-&gt; dbID</td><td>The database ID of the database.</td></tr><tr><td>-&gt; mode</td><td>Which mode to open database in (see “<a href="#">Open Mode Constants</a>”).</td></tr></table>	-> cardNo	Card number database resides on.	-> dbID	The database ID of the database.	-> mode	Which mode to open database in (see “ <a href="#">Open Mode Constants</a> ”).
-> cardNo	Card number database resides on.						
-> dbID	The database ID of the database.						
-> mode	Which mode to open database in (see “ <a href="#">Open Mode Constants</a> ”).						
<b>Result</b>	<code>DmOpenRef</code> to open database. May display a fatal error message if the database parameter is NULL. On all other errors, this function returns 0 and <a href="#">DmGetLastErr</a> returns an error code indicating the reason for failure.						
<b>Comments</b>	<p>Call this routine to open a database for reading or writing, while ignoring any overlay databases that might be associated with it.</p> <p>This routine returns a <code>DmOpenRef</code> which must be used to access particular records in a database. If unsuccessful, 0 is returned and the cause of the error can be determined by calling <a href="#">DmGetLastErr</a>.</p>						
<b>Compatibility</b>	Implemented only if <a href="#">3.5 New Feature Set</a> is present.						
<b>See Also</b>	<a href="#">DmCloseDatabase</a> , <a href="#">DmCreateDatabase</a> , <a href="#">DmFindDatabase</a> , <a href="#">DmOpenDatabaseByTypeCreator</a> , <a href="#">DmDeleteDatabase</a> , <a href="#">DmOpenDatabase</a>						

## Data and Resource Manager

### *Data Manager Functions*

---

## DmPositionInCategory

**Purpose** Return a position of a record within the specified category.

**Declared In** DataMgr.h

**Prototype** `UInt16 DmPositionInCategory (DmOpenRef dbP,  
                  UInt16 index, UInt16 category)`

**Parameters**

-> dbP	DmOpenRef to open database.
-> index	Index of the record.
-> category	Index of category to search.

**Result** Returns the position (zero-based). If the specified index is out of range, this function returns 0 and [DmGetLastErr](#) returns an error code indicating the reason for failure. Note that this means a 0 return value might indicate either success or failure. If this function returns 0 and [DmGetLastErr](#) returns errNone, the return value indicates that this is the first record in the category.

**Comments** Because this function must examine all records up to the current record, it can be slow to return, especially when called on a large database.

Records that have the deleted bit set are ignored, and if the user has specified that private records should be hidden or masked, private records are ignored as well.

If the record is ROM-based (pointer accessed) this routine makes a fake handle to it and stores this handle in the [DmAccessType](#) structure.

To learn which category a record is in, use [DmRecordInfo](#).

**See Also** [DmQueryNextInCategory](#), [DmSeekRecordInCategory](#),  
[DmMoveCategory](#)

## DmQueryNextInCategory

<b>Purpose</b>	Return a handle to the next record in the specified category for reading only (does not set the busy bit).							
<b>Declared In</b>	DataMgr.h							
<b>Prototype</b>	<code>MemHandle DmQueryNextInCategory (DmOpenRef dbP,                   UInt16 *indexP, UInt16 category)</code>							
<b>Parameters</b>	<table><tr><td>-&gt; dbP</td><td>DmOpenRef to open database.</td></tr><tr><td>&lt;-&gt; indexP</td><td>Index of a known record (often retrieved with <a href="#">DmPositionInCategory</a>). If a “next” record is found, this index is updated to indicate that record.</td></tr><tr><td>-&gt; category</td><td>Index of category to query, or dmAllCategories to find the next record in any category.</td></tr></table>		-> dbP	DmOpenRef to open database.	<-> indexP	Index of a known record (often retrieved with <a href="#">DmPositionInCategory</a> ). If a “next” record is found, this index is updated to indicate that record.	-> category	Index of category to query, or dmAllCategories to find the next record in any category.
-> dbP	DmOpenRef to open database.							
<-> indexP	Index of a known record (often retrieved with <a href="#">DmPositionInCategory</a> ). If a “next” record is found, this index is updated to indicate that record.							
-> category	Index of category to query, or dmAllCategories to find the next record in any category.							
<b>Result</b>	Returns a handle to the record following a known record, along with the index of that record. If a record couldn't be found, this function returns NULL, and <a href="#">DmGetLastErr</a> returns an error code indicating the reason for failure.							
<b>Comments</b>	<p>This function begins searching the database from the record at *indexP for a record that is in the specified category. If the record at *indexP belongs to that category, then a handle to it is returned. If not, the function continues searching until it finds a record in the category.</p> <p>Records that have the deleted bit set are skipped, and if the user has specified that private records should be hidden or masked, private records are skipped as well.</p> <p>Thus, if you want to find the next record in the category after the one you have an index for, increment the index value before calling this function. For example:</p>							

```
DmOpenRef myDB;  
UInt16 record, attr, category, pos;
```

## Data and Resource Manager

### Data Manager Functions

---

```
MemHandle newRech;

DmRecordInfo(myDB, record, &attr, NULL, NULL);
category = attr & dmRecAttrCategoryMask;
pos = DmPositionInCategory(myDB, record,
                           category);
pos++;
newRech = DmQueryNextInCategory(myDB, &pos,
                                 category);
```

**See Also** [DmNumRecordsInCategory](#), [DmPositionInCategory](#),  
[DmSeekRecordInCategory](#)

## DmQueryRecord

<b>Purpose</b>	Return a handle to a record for reading only (does not set the busy bit).
<b>Declared In</b>	DataMgr.h
<b>Prototype</b>	MemHandle DmQueryRecord (DmOpenRef dbP, UInt16 index)
<b>Parameters</b>	-> dbP                    DmOpenRef to open database. -> index                Which record to retrieve.
<b>Result</b>	Returns a record handle. If an error occurs, this function returns NULL, and <a href="#">DmGetLastErr</a> returns an error code indicating the reason for failure.  Some releases may display a fatal error message if the specified index is out of range.
<b>Comments</b>	Returns a handle to the given record. Use this routine only when viewing the record. This routine successfully returns a handle to the record even if the record is busy.  If the record is ROM-based (pointer accessed) this routine returns the fake handle to it.

## DmQuickSort

**Purpose** Sort records in a database.

**Declared In** DataMgr.h

**Prototype** Err DmQuickSort (DmOpenRef dbP, DmComparF \*compar, Int16 other)

**Parameters**

-> dbP	Database access pointer.
-> compar	Comparison function. See <a href="#">DmComparF</a> .
-> other	Any value the application wants to pass to the comparison function. This parameter is often used to indicate a sort direction (ascending or descending).

**Result** Returns errNone if no error, or one of the following if an error occurs:

- [dmErrReadOnly](#)
- [dmErrNotRecordDB](#)

Some releases may display a fatal error message instead of returning the error code.

**Comments** Deleted records are placed last in any order. All others are sorted according to the passed comparison function.

After DmQuickSort returns, equal database records do not have a consistent order. That is, if DmQuickSort is passed two equal records, their resulting order is unpredictable. To prevent records that contain the same data from being rearranged in an unpredictable order, pass the record's unique ID to the comparison function (using the [SortRecordInfoType](#) structure).

DmQuickSort contains its own stack to limit uncontrolled recursion. When the stack is full DmQuickSort instead performs an insertion sort. An insertion sort is also performed when the number of records is low, avoiding the noticeable overhead of a quick sort with a small number of records. Finally, if the records seem mostly

## Data and Resource Manager

### Data Manager Functions

---

sorted an insertion sort is performed to move only those records that need moving.

**See Also** [DmFindSortPositionV10](#), [DmInsertionSort](#)

## DmRecordInfo

**Purpose** Retrieve the record information as stored in the database header.

**Declared In** DataMgr.h

**Prototype** Err DmRecordInfo (DmOpenRef dbP, UInt16 index,  
UInt16 \*attrP, UInt32 \*uniqueIDP,  
LocalID \*chunkIDP)

<b>Parameters</b>	-> dbP	DmOpenRef to open database.
	-> index	Index of the record.
	<- attrP	The record's attributes. See “ <a href="#">Record Attribute Constants</a> .” Pass NULL for this parameter if you don't want to retrieve this value.
	<- uniqueIDP	The record's unique ID. Pass NULL for this parameter if you don't want to retrieve this value.
	<- chunkIDP	The record's local ID. Pass NULL for this parameter if you don't want to retrieve this value.

**Result** Returns errNone if no error or [dmErrIndexOutOfRange](#) if the specified record can't be found. Some releases may display a fatal error message instead of returning the error code.

**See Also** [DmNumRecords](#), [DmSetRecordInfo](#), [DmQueryNextInCategory](#)

## DmReleaseRecord

<b>Purpose</b>	Clear the busy bit for the given record and set the dirty bit if dirty is true.
<b>Declared In</b>	DataMgr.h
<b>Prototype</b>	Err DmReleaseRecord (DmOpenRef dbP, UInt16 index, Boolean dirty)
<b>Parameters</b>	-> dbP                    DmOpenRef to open database. -> index                 The record to unlock. -> dirty                  If true, set the dirty bit.
<b>Result</b>	Returns errNone if no error, or <a href="#">dmErrIndexOutOfRange</a> if the specified index is out of range. Some releases may display a fatal error message instead of returning the error code.
<b>Comments</b>	Call this routine when you finish modifying or reading a record that you've called <a href="#">DmGetRecord</a> on or created using <a href="#">DmNewRecord</a> .
<b>See Also</b>	<a href="#">DmGetRecord</a>

## DmReleaseResource

<b>Purpose</b>	Release a resource acquired with <a href="#">DmGetResource</a> .
<b>Declared In</b>	DataMgr.h
<b>Prototype</b>	Err DmReleaseResource (MemHandle resourceH)
<b>Parameters</b>	-> resourceH         Handle to resource.
<b>Result</b>	Returns errNone if no error.

## Data and Resource Manager

### Data Manager Functions

---

**Comments** Marks a resource as being no longer needed by the application.

**See Also** [DmGet1Resource](#), [DmGetResource](#)

## DmRemoveRecord

**Purpose** Remove a record from a database and dispose of its data chunk.

**Declared In** DataMgr.h

**Prototype** Err DmRemoveRecord (DmOpenRef dbP, UInt16 index)

**Parameters** -> dbP DmOpenRef to open database.  
-> index Index of the record to remove.

**Result** Returns errNone if no error, or one of the following if an error occurs:

- [dmErrReadOnly](#)
- [dmErrIndexOutOfRange](#)
- [dmErrNotRecordDB](#)
- [memErrChunkLocked](#)
- [memErrInvalidParam](#)

Some releases may display a fatal error message instead of returning the error code.

**Comments** Disposes of a the record's data chunk and removes the record's entry from the database header. DmRemoveRecord should only be used for newly-created records that have just been deleted or records that have never been sync'ed.

**See Also** [DmDetachRecord](#), [DmDeleteRecord](#), [DmArchiveRecord](#), [DmNewRecord](#)

## DmRemoveResource

**Purpose** Delete a resource from a resource database.

**Declared In** DataMgr.h

**Prototype** Err DmRemoveResource (DmOpenRef dbP, UInt16 index)

**Parameters**

-> dbP	DmOpenRef to open database.
-> index	Index of resource to delete.

**Result** Returns errNone if no error, or one of the following if an error occurs:

- [dmErrCorruptDatabase](#)
- [dmErrIndexOutOfRange](#)
- [dmErrReadOnly](#)
- [memErrChunkLocked](#)
- [memErrInvalidParam](#)
- [memErrNotEnoughSpace](#)

May display a fatal error message if the database is not a resource database.

**Comments** This routine disposes of the memory manager chunk that holds the given resource and removes its entry from the database header.

**See Also** [DmDetachResource](#), [DmRemoveResource](#), [DmAttachResource](#)

## Data and Resource Manager

### *Data Manager Functions*

---

## DmRemoveSecretRecords

**Purpose** Remove all secret records.

**Declared In** DataMgr.h

**Prototype** Err DmRemoveSecretRecords (DmOpenRef dbP)

**Parameters** -> dbP DmOpenRef to open database.

**Result** Returns errNone if no error, or one of the following if an error occurs:

- [dmErrReadOnly](#)
- [dmErrNotRecordDB](#)

Some releases may display a fatal error message instead of returning the error code.

**See Also** [DmRemoveRecord](#), [DmRecordInfo](#), [DmSetRecordInfo](#)

## DmResizeRecord

**Purpose** Resize a record by index.

**Declared In** DataMgr.h

**Prototype** MemHandle DmResizeRecord (DmOpenRef dbP,  
UInt16 index, UInt32 newSize)

**Parameters** -> dbP DmOpenRef to open database.

-> index Which record to retrieve.

-> newSize New size of record.

**Result** Handle to resized record. Returns NULL if there is not enough space to resize the record, and [DmGetLastErr](#) returns an error code indicating the reason for failure. Some releases may display a fatal error message instead of returning the error code.

**Comments** This routine reallocates the record in another heap of the same memory card if the current heap is not big enough. If this happens, the handle changes, so be sure to use the returned handle to access the resized record.

## DmResizeResource

**Purpose** Resize a resource and return the new handle.

**Declared In** DataMgr.h

**Prototype** MemHandle DmResizeResource (MemHandle resourceH,  
UInt32 newSize)

**Parameters** -> resourceH Handle to resource.  
-> newSize Desired new size of resource.

**Result** Returns a handle to newly sized resource. Returns NULL if there is not enough space to resize the resource, and [DmGetLastErr](#) returns an error code indicating the reason for failure. Some releases may display a fatal error message instead of returning the error code.

**Comments** Resizes the resource and returns a new handle. If necessary in order to grow the resource, this routine will reallocate it in another heap on the same memory card that it is currently in.

The handle may change if the resource had to be reallocated in a different data heap because there was not enough space in its present data heap.

## Data and Resource Manager

### *Data Manager Functions*

---

## DmResourceInfo

**Purpose** Retrieve information on a given resource.

**Declared In** DataMgr.h

**Prototype** Err DmResourceInfo (DmOpenRef dbP, UInt16 index,  
DmResType \*resTypeP, DmResID \*resIDP,  
LocalID \*chunkLocalIDP)

**Parameters**

-> dbP	DmOpenRef to open database.
-> index	Index of resource to get info on.
<- resTypeP	The resource type. Pass NULL if you don't want to retrieve this information.
<- resIDP	The resource ID. Pass NULL if you don't want to retrieve this information.
<- chunkLocalIDP	The memory manager local ID of the resource data. Pass NULL if you don't want to retrieve this information.

**Result** Returns errNone if no error or [dmErrIndexOutOfRange](#) if an error occurred. May display a fatal error message if the database is not a resource database.

**Comments** If dbP is a pointer to a base resource database, the information returned is about the resource from that database alone; this function ignores any associated overlay.

**See Also** [DmGetResource](#), [DmGet1Resource](#), [DmSetResourceInfo](#),  
[DmFindResource](#), [DmFindResourceType](#)

## DmSearchRecord

**Purpose** Search all open record databases for a record with the handle passed.

**Declared In** DataMgr.h

**Prototype** UInt16 DmSearchRecord (MemHandle recH,  
DmOpenRef \*dbPP)

**Parameters** -> recH Record handle.  
-< dbPP The database that contains the record recH.

**Result** Returns the index of the record and database access pointer; if not found, returns -1 and \*dbPP is 0.

**See Also** [DmGetRecord](#), [DmFindRecordByID](#), [DmRecordInfo](#)

## DmSearchResource

**Purpose** Search all open resource databases for a resource by type and ID, or by pointer if it is non-NULL.

**Declared In** DataMgr.h

**Prototype** UInt16 DmSearchResource (DmResType resType,  
DmResID resID, MemHandle resH, DmOpenRef \*dbPP)

**Parameters** -> resType Type of resource to search for.  
-> resID ID of resource to search for.  
-> resH Pointer to locked resource, or NULL.  
-< dbPP The resource database that contains the specified resource.

**Result** Returns the index of the resource, stores DmOpenRef in dbPP.

## Data and Resource Manager

### *Data Manager Functions*

---

- Comments** This routine can be used to find a resource in all open resource databases by type and ID or by pointer. If `resH` is NULL, the resource is searched for by type and ID. If `resH` is not NULL, `resType` and `resID` is ignored and the index of the resource handle is returned. On return, `*dbPP` contains the access pointer of the resource database that the resource was eventually found in. Once the index of a resource is determined, it can be locked down and accessed by calling [DmGetResourceIndex](#).
- If any of the open databases are overlaid, this function finds and returns the localized version of the resource when searching by type and creator. In this case, the `dbPP` return value is a pointer to the overlay database, not the base resource database.
- See Also** [DmGetResource](#), [DmFindResourceType](#), [DmResourceInfo](#), [DmFindResource](#)

## DmSeekRecordInCategory

<b>Purpose</b>	Return the index of the record nearest the offset from the passed record index whose category matches the passed category. (The <code>offset</code> parameter indicates the number of records to move forward or backward.)											
<b>Declared In</b>	<code>DataMgr.h</code>											
<b>Prototype</b>	<code>Err DmSeekRecordInCategory (DmOpenRef dbP,                   UInt16 *indexP, UInt16 offset, Int16 direction,                   UInt16 category)</code>											
<b>Parameters</b>	<table><tr><td>-&gt; <code>dbP</code></td><td>DmOpenRef to open database.</td></tr><tr><td>&lt;-&gt; <code>index</code></td><td>The index to start the search at. Upon return, contains the index of the record at <code>offset</code> from the index that you passed in.</td></tr><tr><td>-&gt; <code>offset</code></td><td>Offset of the passed record index. This must be a positive number; use <code>dmSeekBackward</code> for the <code>direction</code> parameter to search backwards.</td></tr><tr><td>-&gt; <code>direction</code></td><td>Must be either <code>dmSeekForward</code> or <code>dmSeekBackward</code>.</td></tr><tr><td>-&gt; <code>category</code></td><td>Category index.</td></tr></table>		-> <code>dbP</code>	DmOpenRef to open database.	<-> <code>index</code>	The index to start the search at. Upon return, contains the index of the record at <code>offset</code> from the index that you passed in.	-> <code>offset</code>	Offset of the passed record index. This must be a positive number; use <code>dmSeekBackward</code> for the <code>direction</code> parameter to search backwards.	-> <code>direction</code>	Must be either <code>dmSeekForward</code> or <code>dmSeekBackward</code> .	-> <code>category</code>	Category index.
-> <code>dbP</code>	DmOpenRef to open database.											
<-> <code>index</code>	The index to start the search at. Upon return, contains the index of the record at <code>offset</code> from the index that you passed in.											
-> <code>offset</code>	Offset of the passed record index. This must be a positive number; use <code>dmSeekBackward</code> for the <code>direction</code> parameter to search backwards.											
-> <code>direction</code>	Must be either <code>dmSeekForward</code> or <code>dmSeekBackward</code> .											
-> <code>category</code>	Category index.											
<b>Result</b>	Returns <code>errNone</code> if no error; returns <a href="#"><u>dmErrIndexOutOfRange</u></a> or <a href="#"><u>dmErrSeekFailed</u></a> if an error occurred.											
<b>Comments</b>	<p><code>DmSeekRecordInCategory</code> searches for a record in the specified category. The search begins with the record at <code>index</code>. When it finds a record in the specified category, it decrements the <code>offset</code> parameter and continues searching until a match is found and <code>offset</code> is 0.</p> <p>Because of this, if you use <code>DmSeekRecordInCategory</code> to find the nearest matching record in a particular category, you must pass different <code>offset</code> parameters if the starting record is in the category than if it isn't. If the record at <code>index</code> is in the category, then you</p>											

## Data and Resource Manager

### Data Manager Functions

---

must pass an offset of 1 to find the next record in the category because the comparison is performed before the index value changes. If the record at index isn't in the category, you must pass an offset of 0 to find the next record in the category. In this case, an offset of 1 skips the first matching record.

Records that have the deleted bit set are ignored, and if the user has specified that private records should be hidden or masked, private records are ignored as well.

**See Also** [DmNumRecordsInCategory](#), [DmQueryNextInCategory](#),  
[DmPositionInCategory](#), [DmMoveCategory](#)

## DmSet

**Purpose** Write a specified value into a section of a record. This function also checks the validity of the pointer for the record and makes sure the writing of the record information doesn't exceed the bounds of the record.

**Declared In** DataMgr.h

**Prototype** Err DmSet (void \*recordP, UInt32 offset,  
UInt32 bytes, UInt8 value)

**Parameters**

-> recordP	Pointer to locked data record (chunk pointer).
-> offset	Offset within record to start writing.
-> bytes	Number of bytes to write.
-> value	Byte value to write.

**Result** Returns errNone if no error. May display a fatal error message if the record pointer is invalid or the function overwrites the record.

**Comments** Must be used to write to data manager records because the data storage area is write-protected.

**See Also** [DmWrite](#)

## DmSetDatabaseInfo

<b>Purpose</b>	Set information about a database.																
<b>Declared In</b>	<code>DataMgr.h</code>																
<b>Prototype</b>	<pre>Err DmSetDatabaseInfo (UInt16 cardNo, LocalID dbID, const Char *nameP, UInt16 *attributesP, UInt16 *versionP, UInt32 *crDateP, UInt32 *modDateP, UInt32 *bckUpDateP, UInt32 *modNumP, LocalID *appInfoIDP, LocalID *sortInfoIDP, UInt32 *typeP, UInt32 *creatorP)</pre>																
<b>Parameters</b>	<table><tr><td>-&gt; cardNo</td><td>Card number the database resides on.</td></tr><tr><td>-&gt; dbID</td><td>Database ID of the database.</td></tr><tr><td>-&gt; nameP</td><td>Pointer to the new name of the database, or NULL. A database name can be up to 32 ASCII bytes long, including the null terminator (as specified by <code>dmDBNameLength</code>). Database names must use only 7-bit ASCII characters (0x20 through 0x7E).</td></tr><tr><td>-&gt; attributesP</td><td>Pointer to new attributes variable, or NULL. See “<a href="#">Database Attribute Constants</a>” for a list of possible values.</td></tr><tr><td>-&gt; versionP</td><td>Pointer to new version, or NULL.</td></tr><tr><td>-&gt; crDateP</td><td>Pointer to new creation date variable, or NULL. Specify the value as a number of seconds since Jan. 1, 1904.</td></tr><tr><td>-&gt; modDateP</td><td>Pointer to new modification date variable, or NULL. Specify the value as a number of seconds since Jan. 1, 1904.</td></tr><tr><td>-&gt; bckUpDateP</td><td>Pointer to new backup date variable, or NULL. Specify the value as a number of seconds since Jan. 1, 1904.</td></tr></table>	-> cardNo	Card number the database resides on.	-> dbID	Database ID of the database.	-> nameP	Pointer to the new name of the database, or NULL. A database name can be up to 32 ASCII bytes long, including the null terminator (as specified by <code>dmDBNameLength</code> ). Database names must use only 7-bit ASCII characters (0x20 through 0x7E).	-> attributesP	Pointer to new attributes variable, or NULL. See “ <a href="#">Database Attribute Constants</a> ” for a list of possible values.	-> versionP	Pointer to new version, or NULL.	-> crDateP	Pointer to new creation date variable, or NULL. Specify the value as a number of seconds since Jan. 1, 1904.	-> modDateP	Pointer to new modification date variable, or NULL. Specify the value as a number of seconds since Jan. 1, 1904.	-> bckUpDateP	Pointer to new backup date variable, or NULL. Specify the value as a number of seconds since Jan. 1, 1904.
-> cardNo	Card number the database resides on.																
-> dbID	Database ID of the database.																
-> nameP	Pointer to the new name of the database, or NULL. A database name can be up to 32 ASCII bytes long, including the null terminator (as specified by <code>dmDBNameLength</code> ). Database names must use only 7-bit ASCII characters (0x20 through 0x7E).																
-> attributesP	Pointer to new attributes variable, or NULL. See “ <a href="#">Database Attribute Constants</a> ” for a list of possible values.																
-> versionP	Pointer to new version, or NULL.																
-> crDateP	Pointer to new creation date variable, or NULL. Specify the value as a number of seconds since Jan. 1, 1904.																
-> modDateP	Pointer to new modification date variable, or NULL. Specify the value as a number of seconds since Jan. 1, 1904.																
-> bckUpDateP	Pointer to new backup date variable, or NULL. Specify the value as a number of seconds since Jan. 1, 1904.																

## Data and Resource Manager

### Data Manager Functions

---

-> modNumP	Pointer to new modification number variable, or NULL.
-> appInfoIDP	Pointer to new appInfoID variable, or NULL.
-> sortInfoIDP	Pointer to new sortInfoID variable, or NULL.
-> typeP	Pointer to new type variable, or NULL.
-> creatorP	Pointer to new creator variable, or NULL.

**Result** Returns `errNone` if no error or one of the following if an error occurred:

- [dmErrInvalidDatabaseName](#)
- [dmErrAlreadyExists](#)
- [dmErrInvalidParam](#)

**Comments** When this call changes `appInfoID` or `sortInfoID`, the old chunk ID (if any) is marked as an orphaned chunk<sup>1</sup> and the new chunk ID is unorphaned. Consequently, you shouldn't replace an existing `appInfoID` or `sortInfoID` if that chunk has already been attached to another database.

Call this routine to set any or all information about a database except for the card number and database ID. This routine sets the new value for any non-NULL parameter.

**See Also** [DmDatabaseInfo](#), [DmOpenDatabaseInfo](#), [DmFindDatabase](#), [DmGetNextDatabaseByTypeCreator](#), [TimDateTimeToSeconds](#)

---

1. An “orphaned chunk” is one that is allocated in the storage heap, but to which nothing refers. If the orphaned chunk is not put into a database as a record, an `appInfo` block, or the like, and if the application doesn’t keep track of it—in a global variable, perhaps—it could get lost. If the application doesn’t get around to freeing the chunk before it quits or crashes, or before the device is reset, that storage will be forever unusable: the user can’t delete it since the user only deletes databases.

During a soft reset, the OS walks through the storage heap and frees any orphaned chunks that it finds. Since most users reset only rarely, however, you shouldn’t rely on this happening.

## DmSetRecordInfo

<b>Purpose</b>	Set record information stored in the database header.								
<b>Declared In</b>	<code>DataMgr.h</code>								
<b>Prototype</b>	<code>Err DmSetRecordInfo (DmOpenRef dbP, UInt16 index, UInt16 *attrP, UInt32 *uniqueIDP)</code>								
<b>Parameters</b>	<table><tr><td>-&gt; dbP</td><td>DmOpenRef to open database.</td></tr><tr><td>-&gt; index</td><td>Index of record.</td></tr><tr><td>-&gt; attrP</td><td>Pointer to new attribute variable, or NULL. See “<a href="#">Record Attribute Constants</a>” for a list of possible values.</td></tr><tr><td>-&gt; uniqueIDP</td><td>Pointer to new unique ID variable, or NULL.</td></tr></table>	-> dbP	DmOpenRef to open database.	-> index	Index of record.	-> attrP	Pointer to new attribute variable, or NULL. See “ <a href="#">Record Attribute Constants</a> ” for a list of possible values.	-> uniqueIDP	Pointer to new unique ID variable, or NULL.
-> dbP	DmOpenRef to open database.								
-> index	Index of record.								
-> attrP	Pointer to new attribute variable, or NULL. See “ <a href="#">Record Attribute Constants</a> ” for a list of possible values.								
-> uniqueIDP	Pointer to new unique ID variable, or NULL.								
<b>Result</b>	Returns <code>errNone</code> if no error, or one of the following if an error occurred: <ul style="list-style-type: none"><li>• <a href="#">dmErrReadOnly</a></li><li>• <a href="#">dmErrNotRecordDB</a></li><li>• <a href="#">dmErrIndexOutOfRange</a></li></ul> Some releases may display a fatal error message instead of returning the error code.								
<b>Comments</b>	Sets information about a record. This routine cannot be used to set the <code>dmRecAttrBusy</code> bit; instead, use <a href="#">DmGetRecord</a> to set the bit and <a href="#">DmReleaseRecord</a> to clear it.  Normally, the unique ID for a record is automatically created by the data manager when a record is created using <a href="#">DmNewRecord</a> , so an application would not typically change the unique ID.								
<b>See Also</b>	<a href="#">DmNumRecords</a> , <a href="#">DmRecordInfo</a>								

## Data and Resource Manager

### *Data Manager Functions*

---

## DmSetResourceInfo

**Purpose** Set information on a given resource.

**Declared In** DataMgr.h

**Prototype** Err DmSetResourceInfo (DmOpenRef dbP,  
                  UInt16 index, DmResType \*resTypeP,  
                  DmResID \*resIDP)

**Parameters**

-> dbP	DmOpenRef to open database.
-> index	Index of resource to set info for.
-> resTypeP	Pointer to new resType (resource type), or NULL.
-> resIDP	Pointer to new resource ID, or NULL.

**Result** Returns errNone if no error, or one of the following if an error occurred:

- [dmErrIndexOutOfRange](#)
- [dmErrReadOnly](#)

May display a fatal error message if the database is not a resource database.

**Comments** Use this routine to set all or a portion of the information on a particular resource. Any or all of the new info pointers can be NULL. If not NULL, the type and ID of the resource are changed to \*resTypeP and \*resIDP.

## DmStrCopy

**Purpose** Copies a string to a record within a database that is open for writing.

**Declared In** DataMgr.h

**Prototype** Err DmStrCopy (void \*recordP, UInt32 offset,  
const Char \*srcP)

## Data and Resource Manager

### Data Manager Functions

---

<b>Parameters</b>	<-> recordP -> offset -> srcP	Pointer to data record (chunk pointer). Offset within record to start writing. Pointer to null-terminated string.
<b>Result</b>	Returns errNone if no error. May display a fatal error message if the record pointer is invalid or the function overwrites the record.	
<b>Comments</b>	This is one of the routines that must be used to write to Data Manager records; because the data storage area is write-protected, you cannot write to it directly. This routine checks the validity of the chunk pointer for the record to insure that writing the record will not exceed the chunk bounds. DmStrCopy is a convenience method that determines the size of the supplied string and then simply calls <a href="#">DmWrite</a> .	
<b>See Also</b>	<a href="#">DmSet</a>	

## DmWrite

<b>Purpose</b>	Copies a specified number of bytes to a record within a database that is open for writing.
<b>Declared In</b>	DataMgr.h
<b>Prototype</b>	Err DmWrite (void *recordP, UInt32 offset, const void *srcP, UInt32 bytes)
<b>Parameters</b>	<-> recordP -> offset -> srcP -> bytes
	Pointer to locked data record (chunk pointer). Offset within record to start writing. Pointer to data to copy into record. Number of bytes to write.
<b>Result</b>	Returns errNone if no error. May display a fatal error message if the record pointer is invalid or the function overwrites the record.

**Comments** This is one of the routines that must be used to write to Data Manager records; because the data storage area is write-protected, you cannot write to it directly. This routine checks the validity of the chunk pointer for the record to insure that writing the record will not exceed the chunk bounds.

**See Also** [DmStrCopy](#), [DmSet](#)

## DmWriteCheck

**Purpose** Check the parameters of a write operation to a data storage chunk before actually performing the write.

**Declared In** DataMgr.h

**Prototype** Err DmWriteCheck (void \*recordP, UInt32 offset,  
UInt32 bytes)

**Parameters** -> recordP Locked pointer to recordH.  
-> offset Offset into record to start writing.  
-> bytes Number of bytes to write.

**Result** Returns errNone if no error; returns [dmErrNotValidRecord](#) or [dmErrWriteOutOfBounds](#) if an error occurred.

# Application-Defined Functions

## DmComparF

**Purpose** Compares two records in a database.

**Declared In** DataMgr.h

**Prototype**

```
typedef Int16 DmComparF (void *rec1, void *rec2,
Int16 other, SortRecordInfoPtr rec1SortInfo,
SortRecordInfoPtr rec2SortInfo,
MemHandle appInfoH)
```

**Parameters**

-> rec1, rec2	Pointers to the two records to compare.
-> other	Any other custom information you want passed to the comparison function. This parameter is often used to indicate a sort direction (ascending or descending).
-> rec1SortInfo, rec2SortInfo	Pointers to <a href="#">SortRecordInfoType</a> structures that specify unique sorting information for the two records.
-> appInfoH	A handle to the database's application info block.

**Result** Returns:

- 0 if rec1 = rec2.
- < 0 if rec1 < rec2.
- > 0 if rec1 > rec2.

**Comments** This function is used to sort the records in a database. It is specifically called by [DmFindSortPosition](#), [DmInsertionSort](#), and [DmQuickSort](#).

**Compatibility** The DmComparF prototype changed in Palm OS version 2.0. Previously, the prototype only defined the first three parameters.

As a rule, the change in the number of arguments from three to six doesn't cause problems when a 1.0 application is run on a 2.0 device because the system only pulls the arguments from the stack that are there.

Keep in mind, however, that some optimized applications built with tools other than Metrowerks CodeWarrior for Palm OS may have problems as a result of the change in arguments when running on a 2.0 or later device.

## **Data and Resource Manager**

### *Application-Defined Functions*

---

# Error Manager

---

This chapter provides reference material for the error manager. The error manager API is declared in the header files `ErrorMgr.h` and `ErrorBase.h`. This chapter covers:

- [ERROR\\_CHECK\\_LEVEL Define](#)
- [Error Manager Data Structures](#)
- [Error Manager Functions](#)

For more information on the error manager, see the chapter “[Debugging Strategies](#)” in the *Palm OS Programmer’s Companion*, vol. I.

## **ERROR\_CHECK\_LEVEL Define**

The error manager uses the compiler define `ERROR_CHECK_LEVEL` to control the level of error messages displayed. You can set the value of the compiler define to control which level of error checking and display is compiled into the application. Three levels of error checking are supported: none, partial, and full.

<b>If you set <code>ERR_CHECK_LEVEL</code> to...</b>	<b>The compiler...</b>
<code>ERROR_CHECK_NONE (0)</code>	Doesn’t compile in any error calls.
<code>ERROR_CHECK_PARTIAL (1)</code>	Compiles in only <code>ErrDisplay</code> and <code>ErrFatalDisplayIf</code> calls.
<code>ERROR_CHECK_FULL (2)</code>	Compiles in all three calls.

# Error Manager Data Structures

## ErrExceptionType

An ErrExceptionType structure is created for each [ErrTry](#) and [ErrCatch](#) block. At any point in the program, there is a linked list of these structures. An ErrExceptionType structure stores information about the state of the machine (register values) at the start of the ErrTry block.

```
typedef struct ErrExceptionType {
    struct ErrExceptionType *nextP;
    ErrJumpBuf state;
    Int32 err;
} ErrExceptionType;
typedef ErrExceptionType *ErrExceptionPtr;
```

### Field Descriptions

nextP    Next ErrExceptionType structure in the linked list.  
state    Storage for setjmp/longjmp.  
err       Error code.

# Error Manager Functions

## ErrAlert

**Purpose**   Macro that displays an alert dialog for runtime errors.

**Declared In**   ErrorBase.h

**Prototype**   ErrAlert (err)

**Parameters**   -> err                  An error code (as type Err).

**Result**   Returns 0, which indicates that the OK button has been clicked to dismiss the dialog.

<b>Comments</b>	<p>This macro is intended for use by applications that are likely to receive runtime errors when the application itself is not at fault. For example, a networking application might use it to display an alert if the remote server cannot be found.</p> <p>The error message displayed on the dialog is stored in a 'tSTL' resource. A 'tSTL' resource contains a list of strings that can be looked up by index. The <code>err</code> parameter is used as the index into this list.</p> <p>To use application-defined error codes in <code>ErrAlert</code>, make sure that all of your error codes are greater than or equal to <code>appErrorClass</code>. This way, the error manager looks up the code in the application's 'tSTL' resource number 0. All other error codes are taken from 'tSTL' resource stored in the system.</p>
<b>Compatibility</b>	Implemented only if <a href="#">3.2 New Feature Set</a> is present.

## ErrCatch

<b>Purpose</b>	Macro that marks the end of an <a href="#">ErrTry</a> block and the beginning of an ErrCatch block.
<b>Declared In</b>	<code>ErrorBase.h</code>
<b>Prototype</b>	<code>ErrCatch (theErr)</code>
<b>Parameters</b>	<code>-&gt; theErr</code> An exception code identifying the reason for the failure. This is the value supplied to the <a href="#">ErrThrow</a> call that caused the jump to this ErrCatch block.
<b>Result</b>	Returns nothing.
<b>Comments</b>	ErrCatch can only be used in conjunction with <a href="#">ErrTry</a> and <a href="#">ErrEndCatch</a> . See the comments under <a href="#">ErrTry</a> for usage instructions. <p><a href="#">ErrTry</a>, <a href="#">ErrCatch</a> and <a href="#">ErrThrow</a> are based on <code>setjmp</code> and <code>longjmp</code>. At the beginning of an <a href="#">ErrTry</a> block, <code>setjmp</code> saves the</p>

## Error Manager

### Error Manager Functions

---

machine registers. ErrThrow calls longjmp, which restores the registers and jumps to the beginning of the ErrCatch block. Therefore, any changes in the ErrTry block to variables stored in registers aren't retained when entering the ErrCatch block.

The solution is to declare variables that you want to use in both the ErrTry and ErrCatch blocks as "volatile". For example:

```
volatile long x = 1; // Declare volatile local variable
ErrTry {
    x = 100; // Set local variable in Try
    ErrThrow(-1);
}
ErrCatch(inErr) {
    if (x > 1) { // Use local variable in Catch
        SysBeep(1);
    }
} ErrEndCatch
```

---

If you have many local variables after the ErrCatch you may want to put the ErrTry and ErrCatch in a separate enclosing function.

## ErrDisplay

**Purpose** Macro that displays an error alert if error checking is set to partial or full.

**Declared In** ErrorMgr.h

**Prototype** ErrDisplay (msg)

**Parameters** -> msg                    Error message text as a string.

**Result** No return value.

**Comments** Call this macro to display an error message, source code filename, and line number. This macro is compiled into the code only if the compiler define ERROR\_CHECK\_LEVEL is set to 1 or 2 (ERROR\_CHECK\_PARTIAL or ERROR\_CHECK\_FULL).

**See Also** [ErrFatalDisplayIf](#), [ErrNonFatalDisplayIf](#)

## ErrDisplayFileLineMsg

**Purpose** Display a dialog with an error message. Do not allow the user to exit the dialog or continue.

**Declared In** ErrorBase.h

**Prototype**

```
void ErrDisplayFileLineMsg  
(const Char *const filename, UInt16 lineno,  
const Char *const msg)
```

**Parameters**

-> filename	Source code filename.
-> lineno	Line number in the source code file.
-> msg	Message to display.

**Result** Never returns.

**Comment** Called by [ErrFatalDisplayIf](#) and [ErrNonFatalDisplayIf](#). This function is useful when the application is already on the device and being tested by users.

On Japanese systems, the system displays a generic message indicating that an error has occurred instead of displaying the English message.

**See Also** [ErrFatalDisplayIf](#), [ErrNonFatalDisplayIf](#), [ErrDisplay](#)

## ErrEndCatch

**Purpose** Macro that marks the end of an [ErrCatch](#) block.

**Declared In** ErrorBase.h

**Prototype** ErrEndCatch

**Parameters** None.

**Result** Returns nothing.

## Error Manager

### Error Manager Functions

---

**Comments** ErrEndCatch can only be used in conjunction with [ErrTry](#) and ErrCatch. See the comments under ErrTry for usage instructions.

## ErrExceptionList

**Purpose** Return the address of the pointer to the first [ErrExceptionType](#) structure in the linked list of exception structures.

**Declared In** ErrorBase.h

**Prototype** MemPtr \*ErrExceptionList (void)

**Parameters** None.

**Result** Returns the address of the pointer to the first ErrExceptionType structure linked into the exception list.

**Comments** This function is used by the ErrTry and ErrCatch macros as well as the [ErrThrow](#) function in order to find the exception list.

When called from an application, this routine returns the application's exception list.

## ErrFatalDisplayIf

**Purpose** Macro that displays an error alert dialog if condition is true and error checking is set to partial or full.

**Declared In** ErrorMgr.h

**Prototype** ErrFatalDisplayIf (condition, msg)

**Parameters** -> condition A boolean value. If true, display the error.  
-> msg Error message text as a string.

**Result** No return value.

**Comments** Call this macro to display a fatal error message, source code filename, and line number. The alert is displayed only if condition is true. The dialog is cleared only when the user resets the system by responding to the dialog.

This macro is compiled into the code if the compiler define ERROR\_CHECK\_LEVEL is set to 1 or 2 (ERROR\_CHECK\_PARTIAL or ERROR\_CHECK\_FULL).

**See Also** [ErrNonFatalDisplayIf](#), [ErrDisplay](#),

## **ErrNonFatalDisplayIf**

**Purpose** Macro that displays an error alert dialog if condition is true and error checking is set to full.

**Declared In** ErrorMgr.h

**Prototype** ErrNonFatalDisplayIf (condition, msg)

**Parameters** -> condition A boolean value. If true, display the error.  
-> msg Error message text as a string.

**Result** No return value.

**Comments** Call this macro to display a nonfatal error message, source code filename, and line number. The alert is displayed only if condition is true. The alert dialog is cleared when the user selects to continue (or resets the system).

This macro is compiled into the code only if the compiler define ERROR\_CHECK\_LEVEL is set to 2 (ERROR\_CHECK\_FULL).

**See Also** [ErrFatalDisplayIf](#), [ErrDisplay](#)

## Error Manager

### Error Manager Functions

---

## ErrThrow

**Purpose** Cause a jump to the nearest Catch block.

**Declared In** ErrorBase.h

**Prototype** void ErrThrow (Int32 err)

**Parameters** -> err                    Error code.

**Result** Never returns.

**Comments** Use the macros ErrTry, ErrCatch, and ErrEndCatch in conjunction with this function.

**See Also** [ErrFatalDisplayIf](#), [ErrNonFatalDisplayIf](#), [ErrDisplay](#)

## ErrTry

**Purpose** Macro that marks the beginning of a try/catch block.

**Declared In** ErrorBase.h

**Prototype** ErrTry

**Parameters** None.

**Result** Returns nothing.

**Comments** An exception raised by a call to [ErrThrow](#)—even from within a nested subroutine—causes program execution to switch to the beginning of the [ErrCatch](#) block. If the end of the block enclosed by ErrTry is encountered without a call to ErrThrow, execution jumps to the line of code following the [ErrEndCatch](#) macro. See “[The Try-and-Catch Mechanism](#)” on page 377 of the *Palm OS Programmer’s Companion*, vol. I for a more thorough description of how this mechanism works.

**Example** You must structure your code exactly as shown here. You can't use ErrTry without ErrCatch and ErrEndCatch, or vice versa.

```
ErrTry {
    // Do something which may fail. Call ErrThrow to signal
    // failure and force a jump to the following ErrCatch
    // block.
}
ErrCatch(inErr) {
    // Recover or cleanup after a failure in the above ErrTry
    // block. "inErr" is an exception code identifying the
    // reason for the failure.

    // Call ErrThrow if you want to jump out to the next
    // ErrCatch block.

    // The code in this block doesn't execute if the above
    // ErrTry block completes without a call to ErrThrow.
} ErrEndCatch
```

---

## Error Manager

### *Error Manager Functions*

---

# Expansion Manager

---

This chapter provides the following information about the Expansion Manager:

- [Expansion Manager Data Structures](#)
- [Expansion Manager Constants](#)
- [Expansion Manager Functions](#)

The header file `ExpansionMgr.h` declares the Expansion Manager API. For more information on the Expansion Manager, see [Chapter 7, “Expansion,”](#) in *Palm OS Programmer’s Companion*, vol. I.

Note that the Expansion Manager is an optional system extension; the functions described in this chapter are implemented only if the [Expansion Manager Feature Set](#) is present.

## Expansion Manager Data Structures

### `ExpCardInfoType`

The `ExpCardInfoType` declaration defines a structure that is passed to [`ExpCardInfo`](#). This structure is used to determine the characteristics of the card loaded in the slot. It is initialized by the underlying slot driver with the following information.

```
typedef struct ExpCardInfoTag {
    UInt32 capabilityFlags;
    Char manufacturerStr [expCardInfoStringMaxLen+1];
    Char productStr [expCardInfoStringMaxLen+1];
    Char deviceClassStr [expCardInfoStringMaxLen+1];
    Char deviceUniqueIDStr [expCardInfoStringMaxLen+1];
} ExpCardInfoType, *ExpCardInfoPtr;
```

### Field Descriptions

capabilityFlags	Describes the capabilities of the card. The following flags are currently supported: <ul style="list-style-type: none"><li>• <code>expCapabilityHasStorage</code> indicates that the card supports reading and (possibly) writing.</li><li>• <code>expCapabilityReadOnly</code> indicates that the card is read only.</li><li>• <code>expCapabilitySerial</code> indicates that the card supports a simple serial interface.</li></ul>
manufacturerStr	Names the manufacturer of the card. For example "Palm" or "Motorola".
productStr	Name of the product. For example "SafeBackup 32MB".
deviceClassStr	Describes the type of card, for example, "Backup" or "Ethernet".
deviceUniqueIDStr	Unique identifier for the product. A serial number for example. This value is set to the empty string if no identifier exists.

## Expansion Manager Constants

### Error Codes

The Expansion Manager defines the following error codes:

Constant	Description
<code>expErrUnsupportedOperation</code>	The operation is unsupported or undefined.
<code>expErrNotEnoughPower</code>	The required power is not available.
<code>expErrCardNotPresent</code>	There is no card present in the given slot.

Constant	Description
expErrInvalidSlotRefNum	The slot reference number is not valid.
expErrSlotDeallocated	The slot reference number is within the valid range, but the slot has been deallocated.
expErrCardNoSectorReadWrite	The card does not support the slot driver block read/write API.
expErrCardReadOnly	The card supports the slot driver block read/write API but the card is read only.
expErrCardBadSector	The card supports the slot driver block read/write API but the sector is bad.
expErrCardProtectedSector	The card supports the slot driver block read/write API but the sector is protected.
expErrNotOpen	The slot driver library has not been opened.
expErrStillOpen	The slot driver library is still open; it may have been opened more than once.
expErrUnimplemented	The call is unimplemented.
expErrEnumerationEmpty	There are no values remaining to enumerate.
expErrIncompatibleAPIVer	The API version of the underlying slot driver is not supported by this version of Expansion Manager.

## Defined Media Types

The following media types are defined by the Expansion Manager. These media types are used with the function [VFSVolumeInfo](#) in the `VolumeInfoType.mediaType` field. The media type is also passed as a parameter to the [VFSRegisterDefaultDirectory](#) and [VFSUnregisterDefaultDirectory](#) functions.

## Expansion Manager

### *Expansion Manager Functions*

---

Constant	Value	Description
expMediaType_Any	'wild'	Matches all media types when looking up a default directory
expMediaType_MemoryStick	'mstk'	Memory stick
expMediaType_CompactFlash	'cfsh'	Compact Flash
expMediaType_SecureDigital	'sdig'	Secure Digital
expMediaType_MultiMediaCard	'mmcd'	MultiMedia Card
expMediaType_SmartMedia	'smed'	SmartMedia
expMediaType_RAMDisk	'ramd'	A RAM disk based media
expMediaType_PoserHost	'pose'	Host file system emulated by the Palm OS® Emulator
expMediaType_MacSim	'PSim'	Host file system emulated by the Mac Simulator

---

## Expansion Manager Functions

### **ExpCardGetSerialPort**

**Purpose** Get a card's serial port creator ID for use in serial access.

**Declared In** ExpansionMgr.h

**Prototype** Err ExpCardGetSerialPort (UInt16 slotRefNum,  
                  UInt32 \*portP)

**Parameters** -> slotRefNum     Slot number of slot to check.

`<- portP` Pointer to UInt32 into which the serial port creator ID is stored.

**Result** Returns the following result codes:

errNone No error

`expErrInvalidSlotRefNum`

The specified slot number is invalid.

## expErrSlotDeallocated

The specified slot number is within the valid range but has been deallocated.

**Compatibility** Implemented only if the [Expansion Manager Feature Set](#) is present.

#### **See Also**

[ExpCardInfo](#)

# ExpCardInfo

**Purpose** Obtains information about a card in a given slot.

**Declared In** ExpansionMgr.h

**Prototype** Err ExpCardInfo(UInt16 slotRefNum,  
ExpCardInfoType \*infoP)

**Parameters**    -> slotRefNum    Slot number.  
                  <- infoP            Pointer to [ExpCardInfoType](#) structure.

**Result** Returns one of the following result codes:

errNone No error

`expErrCardNotPresent`

There is no card present in the specified slot.

`expErrInvalidSlotRefNum`

The slot number is invalid.

`expErrSlotDeallocated`

The slot number is within the valid range but has been deallocated.

**Comments** This function returns information about a card, including whether the card supports secondary storage or is strictly read-only, by filling in the `ExpCardInfoType` structure's fields.

**Compatibility** Implemented only if the [Expansion Manager Feature Set](#) is present.

**See Also** [ExpCardGetSerialPort](#), [ExpCardPresent](#),  
[ExpSlotEnumerate](#)

## ExpCardPresent

**Purpose** Determines if a card is present in the given slot.

**Declared In** ExpansionMgr.h

**Prototype** Err ExpCardPresent (UInt16 slotRefNum)

**Parameters** -> slotRefNum Slot number.

**Result** Returns the following result codes:

`errNone` A card is present in the specified slot.

`expErrCardNotPresent` There is no card present in the specified slot.

`expErrInvalidSlotRefNum` The specified slot number is not valid.

`expErrSlotDeallocated` The specified slot number is within the valid range but has been deallocated.

<b>Comments</b>	The Expansion Manager passes the call through to the appropriate slot driver.
<b>Compatibility</b>	Implemented only if the <a href="#">Expansion Manager Feature Set</a> is present.
<b>See Also</b>	<a href="#">ExpCardInfo</a> , <a href="#">ExpSlotEnumerate</a>

## ExpSlotDriverInstall

<b>Purpose</b>	Installs and initializes a slot driver shared library into the system table.				
<b>Declared In</b>	ExpansionMgr.h				
<b>Prototype</b>	<pre>Err ExpSlotDriverInstall (UInt32 dbCreator,                           UInt16 *slotLibRefNumP)</pre>				
<b>Parameters</b>	<table><tr><td>-&gt; dbCreator</td><td>Database creator code of the slot driver library to be installed.</td></tr><tr><td>-&gt; slotLibRefNumP</td><td>Pointer to variable for returning the library reference number (on failure, sysInvalidRefNum is returned in this variable).</td></tr></table>	-> dbCreator	Database creator code of the slot driver library to be installed.	-> slotLibRefNumP	Pointer to variable for returning the library reference number (on failure, sysInvalidRefNum is returned in this variable).
-> dbCreator	Database creator code of the slot driver library to be installed.				
-> slotLibRefNumP	Pointer to variable for returning the library reference number (on failure, sysInvalidRefNum is returned in this variable).				
<b>Result</b>	Returns errNone if the slot driver is installed correctly. Because this function uses <a href="#">SysLibInstall</a> to install the slot driver shared library, ExpSlotDriverInstall may return any of the error codes that SysLibInstall returns, including sysErrLibNotFound, sysErrNoFreeRAM, and sysErrNoFreeLibSlots. It may also return any error code returned by SlotOpen, the implementation of which is specific to a given device manufacturer.				
<b>Comments</b>	This function is not typically called by applications but can be used to load additional slot drivers after the device has booted. It is called internally by the Expansion Manager to install a slot driver shared library into the library table and initialize it for use. Once installed,				

## **Expansion Manager**

### *Expansion Manager Functions*

---

the slotLibRefNum can be used by other functions to refer to the library.

**Compatibility** Implemented only if the [Expansion Manager Feature Set](#) is present.

**See Also** [ExpSlotDriverRemove](#), [ExpSlotLibFind](#)

## **ExpSlotDriverRemove**

**Purpose** Closes and remove a slot driver shared library from the system table.

**Declared In** ExpansionMgr.h

**Prototype** Err ExpSlotDriverRemove (UInt16 slotLibRefNum)

**Parameters** -> slotLibRefNum  
Slot driver shared library reference number.

**Result** Returns errNone.

**Comments** This function is not typically called by applications but can be used to unload slot drivers associated with external slots. It is called internally by the Expansion Manager to remove the shared library from the system table, and, if appropriate, release the slot allocated to the given slotLibRefNum. Prior to removing the slot driver, it unmounts any volumes associated with the slot.

**Compatibility** Implemented only if the [Expansion Manager Feature Set](#) is present.

**See Also** [ExpSlotDriverInstall](#), [ExpSlotLibFind](#), [SysLibRemove](#)

## ExpSlotEnumerate

**Purpose** Iterates through valid slot numbers.

**Declared In** ExpansionMgr.h

**Prototype** Err ExpSlotEnumerate (UInt16 \*slotRefNumP,  
                  UInt32 \*slotIteratorP)

**Parameters** <- slotRefNumP Reference number of the currently-enumerated slot.

-> slotIteratorP  
Pointer to the index of the last entry enumerated. For the first iteration, initialize this parameter to the constant expIteratorStart. Upon return this references the next entry in the directory. If this is the last entry, this parameter is set to expIteratorStop.

**Result** Returns one of the following result codes:

errNone The slot reference number indicated by slotRefNumP is valid.

expErrEnumerationEmpty  
There are no slots left to enumerate.  
slotRefNumP is set to the slot number of the currently enumerated slot or invalidSlotRefNum if there are no slots.

**Comments** This function iterates through the device's slots. The first time this function is called, set \*slotIteratorP to expIteratorStart to find the initial slot. Once set this value is changed with each subsequent call to this function until it reaches the maximum number of slots, at which point ExpSlotEnumerate sets \*slotIteratorP to expIteratorStop.

**Example** The following is an example of how ExpSlotEnumerate should be used:

## Expansion Manager

### Expansion Manager Functions

---

```
UInt16 slotRefNum;
UInt32 slotIterator = expIteratorStart;
while (slotIterator != expIteratorStop) {
    if ((err = ExpSlotEnumerate(&slotRefNum,
                                &slotIterator)) == errNone) {
        // do something with the slotRefNum
    } else {
        // handle error (by breaking out of the
        // loop, most likely
    }
}
```

---

**Compatibility** Implemented only if the [Expansion Manager Feature Set](#) is present.

**See Also** [ExpCardInfo](#), [ExpCardPresent](#), [ExpSlotDriverRemove](#)

## ExpSlotLibFind

**Purpose** Retrieves the slot driver library reference number for the driver that controls the specified slot.

**Declared In** ExpansionMgr.h

**Prototype** Err ExpSlotLibFind(UInt16 slotRefNum,  
                  UInt16 \*slotLibRefNum)

**Parameters** -> slotRefNum Slot number.

<- slotLibRefNum  
                  Pointer to the reference number for the slot  
                  driver library allocated to the given slot.

**Result** Returns the following result codes:

errNone No error.

expErrInvalidSlotRefNum  
                  The slot number is invalid.

`expErrSlotDeallocated`

The slot number is within the valid range but has been deallocated.

**Comments** This function returns the reference number to the slot driver library loaded for the given `slotRefNum`. This function is used when making calls directly to the slot driver library.

**Compatibility** Implemented only if the [Expansion Manager Feature Set](#) is present.

**See Also** [ExpSlotDriverInstall](#), [ExpSlotDriverRemove](#)

## **Expansion Manager**

*Expansion Manager Functions*

---

# Feature Manager

---

This chapter provides reference material for the feature manager. The feature manager API is declared in the header file `FeatureMgr.h`.

For more information on the feature manager, see the section “[Features](#)” in the *Palm OS Programmer’s Companion*, vol. I.

To learn how to use the predefined Palm OS® features to test for the existence of certain OS features, see the “[Compatibility Guide](#)” appendix.

## Feature Manager Functions

### FtrGet

**Purpose** Get a feature.

**Declared In** `FeatureMgr.h`

**Prototype** `Err FtrGet (UInt32 creator, UInt16 featureNum,  
                  UInt32 *valueP)`

**Parameters**

-> creator	Creator ID, which must be registered with PalmSource, Inc. This is usually the same as the creator ID for the application that owns this feature.
-> featureNum	Feature number of the feature.
<- valueP	Value of the feature is returned here.

**Result** Returns 0 if no error, or `ftrErrNoSuchFeature` if the specified feature number doesn’t exist for the specified creator.

## Feature Manager

### Feature Manager Functions

---

**Comments** The value of the feature is application-dependent.

**See Also** [FtrSet](#)

## FtrGetByIndex

**Purpose** Get a feature by index.

**Declared In** FeatureMgr.h

**Prototype** Err FtrGetByIndex (UInt16 index, Boolean romTable,  
                  UInt32 \*creatorP, UInt16 \*numP, UInt32 \*valueP)

**Parameters**

-> index	Index of feature.
-> romTable	If true, index into ROM table; otherwise, index into RAM table.
<- creatorP	Feature creator is returned here.
<- numP	Feature number is returned here.
<- valueP	Feature value is returned here.

**Result** Returns 0 if no error, or ftrErrNoSuchFeature if the index is out of range.

**Comments** This function is intended for system use only. It is used by shell commands. Most applications don't need it.

Until the caller gets back ftrErrNoSuchFeature, it should pass indices for each table (ROM, RAM) starting at 0 and incrementing. Note that in Palm OS 3.1 and higher, the RAM feature table serves the entire system. At system startup, the values in the ROM feature table are copied into the RAM feature table.

## FtrPtrFree

<b>Purpose</b>	Release memory previous allocated with <a href="#">FtrPtrNew</a> .
<b>Declared In</b>	FeatureMgr.h
<b>Prototype</b>	Err FtrPtrFree (UInt32 creator, UInt16 featureNum)
<b>Parameters</b>	-> creator      The creator ID for the feature. -> featureNum   Feature number of the feature.
<b>Result</b>	Returns 0 if no error, or ftrErrNoSuchFeature if an error occurs.
<b>Comments</b>	This function unregisters the feature before freeing the memory associated with it.
<b>Compatibility</b>	Implemented only if <a href="#">3.1 New Feature Set</a> is present.

## FtrPtrNew

<b>Purpose</b>	Allocate feature memory.
<b>Declared In</b>	FeatureMgr.h
<b>Prototype</b>	Err FtrPtrNew (UInt32 creator, UInt16 featureNum, UInt32 size, void **newPtrP)
<b>Parameters</b>	-> creator      Creator ID, which must be registered with PalmSource, Inc. This is usually the same as the creator ID for the application that owns this feature. -> featureNum   Feature number of the feature. -> size           Size in bytes of the temporary memory to allocate. The maximum chunk size is 64K.

## Feature Manager

### Feature Manager Functions

---

<- newPtrP      Pointer to the memory chunk is returned here.

**Result**      Returns 0 if no error, `memErrInvalidParam` if the value of `size` is 0, or `memErrNotEnoughSpace` if there is not enough space to allocate a chunk of the specified size.

**Comments**      This function allocates a chunk of memory and stores a pointer to that chunk in the feature table. The same pointer is returned in `newPtrP`. The memory chunk remains allocated and locked until the next system reset or until you free the chunk with [FtrPtrFree](#). `FtrPtrNew` is useful if you want quick, efficient access to data that persists from one invocation of the application to the next. `FtrPtrNew` stores values on the storage heap rather than the dynamic heap, where free space is often extremely limited. The disadvantage to using feature memory is that writing to storage memory is slower than writing to dynamic memory.

---

**NOTE:** Starting with Palm OS 3.5 `FtrPtrNew` allows allocating chunks larger than 64k. Do keep in mind standard issues with allocating large chunks of memory: there might not be enough contiguous space, and it can impact system performance.

---

You can obtain the pointer to the chunk using [FtrGet](#). To write to the chunk, you **must** use [DmWrite](#) because the chunk is in the storage heap, not the dynamic heap.

For example, if you allocate a memory chunk in this way:

```
FtrPtrNew(appCreator,  
         myFtrMemFtr, 32, &ftrMem);
```

You can later access that memory and write to it using the following:

```
void* data;  
if (!FtrGet(appCreator,  
         myFtrMemFtr, (UInt32*)&data))  
DmWrite(data, 0, &someVal, sizeof(someVal));
```

**Compatibility**      Implemented only if [3.1 New Feature Set](#) is present.

**See Also**      [FtrPtrResize](#)

## FtrPtrResize

<b>Purpose</b>	Resize feature memory.								
<b>Declared In</b>	FeatureMgr.h								
<b>Prototype</b>	<pre>Err FtrPtrResize (UInt32 creator,                   UInt16 featureNum, UInt32 newSize,                   void **newPtrP)</pre>								
<b>Parameters</b>	<table><tr><td>-&gt; creator</td><td>The creator ID for the feature.</td></tr><tr><td>-&gt; featureNum</td><td>Feature number of the feature.</td></tr><tr><td>-&gt; newSize</td><td>New size in bytes for the chunk.</td></tr><tr><td>&lt;- newPtrP</td><td>Pointer to the memory chunk is returned here.</td></tr></table>	-> creator	The creator ID for the feature.	-> featureNum	Feature number of the feature.	-> newSize	New size in bytes for the chunk.	<- newPtrP	Pointer to the memory chunk is returned here.
-> creator	The creator ID for the feature.								
-> featureNum	Feature number of the feature.								
-> newSize	New size in bytes for the chunk.								
<- newPtrP	Pointer to the memory chunk is returned here.								
<b>Result</b>	Returns 0 if no error, or <code>ftrErrNoSuchFeature</code> if the specified feature number doesn't exist for the specified creator, <code>memErrInvalidParam</code> if <code>newSize</code> is 0, or <code>memErrNotEnoughSpace</code> if there's not enough free space available to allocate a chunk of that size.								
<b>Comments</b>	<p>Use this function to resize a chunk of memory previously allocated by <a href="#">FtrPtrNew</a>.</p> <p>This function may move the chunk to a new location in order to resize it, so it is important to use the pointer returned by this function when accessing the memory chunk. The pointer in the feature table is automatically updated to be the same as the pointer returned by this function.</p> <p>If this function fails, the old memory pointer still exists and its data is unchanged.</p>								
<b>Compatibility</b>	Implemented only if <a href="#">3.1 New Feature Set</a> is present.								
<b>See Also</b>	<a href="#">MemHandleResize</a>								

## Feature Manager

### *Feature Manager Functions*

---

## FtrSet

**Purpose** Set a feature.

**Declared In** FeatureMgr.h

**Prototype** Err FtrSet (UInt32 creator, UInt16 featureNum,  
                  UInt32 newValue)

**Parameters**

-> creator	Creator ID, which must be registered with PalmSource, Inc. This is usually the same as the creator ID for the application that owns this feature.
-> featureNum	Feature number for this feature.
-> newValue	New value.

**Result** Returns 0 if no error, or memErrNotEnoughSpace if the feature table must be resized to add a new feature and no space is available.

**Comments** The value of the feature is application-dependent.  
A feature that you define in this manner remains defined until the next system reset or until you explicitly undefine the feature with [FtrUnregister](#).

**See Also** [FtrGet](#), [FtrPtrNew](#)

## FtrUnregister

**Purpose** Unregister a feature.

**Declared In** FeatureMgr.h

**Prototype** Err FtrUnregister (UInt32 creator,  
                  UInt16 featureNum)

**Parameters**

-> creator	Creator ID for the feature.
------------	-----------------------------

-> featureNum Feature number of the feature.

**Result** Returns 0 if no error, or ftrErrNoSuchFeature if the specified feature number doesn't exist for the specified creator.

## **Feature Manager**

### *Feature Manager Functions*

---

# File Streaming

---

This chapter provides reference material for the file streaming API.

- [File Streaming Constants](#)
- [File Streaming Functions](#)
- [File Streaming Error Codes](#)

The header file `FileStream.h` declares the API that this chapter describes. For more information on file streaming, see the chapter “[Files and Databases](#)” in the *Palm OS Programmer’s Companion*, vol. I.

## File Streaming Constants

### Primary Open Mode Constants

This section lists constants passed in the `openMode` parameter to the [FileOpen](#) function. These constants specify the mode in which a file stream is opened.

For each file stream, you must pass to the [FileOpen](#) function only one of the primary mode selectors listed.

#### Constant Values

<code>fileModeReadOnly</code>	Open for read-only access
<code>fileModeReadWrite</code>	Open/create for read/write access, discarding any previous version of stream
<code>fileModeUpdate</code>	Open/create for read/write, preserving previous version of stream if it exists
<code>fileModeAppend</code>	Open/create for read/write, always writing to the end of the stream

## File Streaming

### *File Streaming Constants*

---

## Secondary Open Mode Constants

You can use the | operator (bitwise inclusive OR) to append to a primary mode selector one or more of the secondary mode selectors listed below.

<code>fileModeDontOverwrite</code>	Prevents <code>fileModeReadWrite</code> from discarding an existing stream having the same name; may only be specified together with <code>fileModeReadWrite</code>
<code>fileModeLeaveOpen</code>	Leave stream open when application quits. Most applications should not use this option.
<code>fileModeExclusive</code>	No other application can open the stream until the application that opened it in this mode closes it.
<code>fileModeAnyTypeCreator</code>	Accept any type/creator when opening or replacing an existing stream. Normally, the <a href="#">FileOpen</a> function opens only streams having the specified creator and type. Setting this option enables the <code>FileOpen</code> function to open streams having a type or creator other than those specified.
<code>fileModeTemporary</code>	Delete the stream automatically when it is closed. For more information, see Comment section of <a href="#">FileOpen</a> function description.

# File Streaming Functions

## FileClearerr

<b>Purpose</b>	Clear I/O error status, end of file error status, and last error.
<b>Declared In</b>	FileStream.h
<b>Prototype</b>	Err FileClearerr (FileHand stream)
<b>Parameters</b>	--> stream Handle to open stream.
<b>Result</b>	0 if no error, or a fileErr code if an error occurs. See the section " <a href="#">File Streaming Error Codes</a> " for more information.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">FileGetLastError</a> , <a href="#">FileRewind</a>

## FileClose

<b>Purpose</b>	Close the file stream and destroy its handle. If the stream was opened with fileModeTemporary, it is deleted upon closing.
<b>Declared In</b>	FileStream.h
<b>Prototype</b>	Err FileClose (FileHand stream)
<b>Parameters</b>	--> stream Handle to open stream.
<b>Result</b>	0 if no error, or a fileErr code if an error occurs. See the section " <a href="#">File Streaming Error Codes</a> " for more information.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## File Streaming

### File Streaming Functions

---

## FileControl

<b>Purpose</b>	Perform the operation specified by the <code>op</code> parameter on the <code>stream</code> file stream.								
<b>Declared In</b>	<code>FileStream.h</code>								
<b>Prototype</b>	<pre>Err FileControl (FileOpEnum op, FileHand stream, void *valueP, Int32 *valueLenP)</pre>								
<b>Parameters</b>	<table><tr><td><code>op</code></td><td>The operation to perform, and its associated formal parameters. See the Comments section for a list of possible values.</td></tr><tr><td><code>--&gt; stream</code></td><td>Open stream handle if required for file stream operation.</td></tr><tr><td><code>&lt;-&gt; valueP</code></td><td>Pointer to value or buffer, as required. This parameter is defined by the selector passed as the value of the <code>op</code> parameter. For details, see the Comments section.</td></tr><tr><td><code>&lt;-&gt; valueLenP</code></td><td>Pointer to value or buffer, as required. This parameter is defined by the selector passed as the value of the <code>op</code> parameter. For details, see the Comments section.</td></tr></table>	<code>op</code>	The operation to perform, and its associated formal parameters. See the Comments section for a list of possible values.	<code>--&gt; stream</code>	Open stream handle if required for file stream operation.	<code>&lt;-&gt; valueP</code>	Pointer to value or buffer, as required. This parameter is defined by the selector passed as the value of the <code>op</code> parameter. For details, see the Comments section.	<code>&lt;-&gt; valueLenP</code>	Pointer to value or buffer, as required. This parameter is defined by the selector passed as the value of the <code>op</code> parameter. For details, see the Comments section.
<code>op</code>	The operation to perform, and its associated formal parameters. See the Comments section for a list of possible values.								
<code>--&gt; stream</code>	Open stream handle if required for file stream operation.								
<code>&lt;-&gt; valueP</code>	Pointer to value or buffer, as required. This parameter is defined by the selector passed as the value of the <code>op</code> parameter. For details, see the Comments section.								
<code>&lt;-&gt; valueLenP</code>	Pointer to value or buffer, as required. This parameter is defined by the selector passed as the value of the <code>op</code> parameter. For details, see the Comments section.								
<b>Result</b>	Returns either a value defined by the selector passed as the argument to the <code>op</code> parameter, or an error code resulting from the requested operation. For details, see the Comments section.								
<b>Comments</b>	Normally, you do not call the <a href="#">FileControl</a> function yourself; it is called for you by most of the other file streaming functions and macros to perform common file streaming operations. You can call <a href="#">FileControl</a> yourself to enable specialized read modes.								

<code>fileOpNone</code>	No-op.
<code>fileOpDestructiveReadMode</code>	<p>Enter destructive read mode, and rewind stream to its beginning. Once in this mode, there is no turning back: stream's contents after closing (or crash) are undefined.</p> <p>Destructive read mode deletes blocks as data are read, thus freeing storage automatically. Once in destructive read mode, you cannot re-use the file stream—the contents of the stream are undefined after it is closed or after a crash.</p> <p>Writing to files opened without write access or those that are in destructive read state is not allowed; thus, you cannot call the <a href="#">FileWrite</a>, <a href="#">FileSeek</a>, or <a href="#">FileTruncate</a> functions on a stream that is in destructive read mode. One exception to this rule applies to streams that were opened in “write + append” mode and then switched into destructive read state. In this case, the <a href="#">FileWrite</a> function can append data to the stream, but it also preserves the current stream position so that subsequent reads pick up where they left off (you can think of this as a pseudo-pipe).</p>
<code>fileOpGetEOFStatus</code>	<p><b>ARGUMENTS:</b></p> <p><code>stream</code> = open stream handle <code>valueP</code> = NULL <code>valueLenP</code> = NULL</p> <p><b>RETURNS:</b></p> <p>zero on success; <code>fileErr...</code> on error</p> <p>Get end-of-file status (like C runtime’s <code>feof</code>) (<code>err = fileErrEOF</code>). Indicates end of file condition. Use <a href="#">FileClearerr</a> to clear this error status.</p>

## File Streaming

### *File Streaming Functions*

---

	<p><b>ARGUMENTS:</b> <code>stream</code> = open stream handle <code>valueP</code> = NULL <code>valueLenP</code> = NULL</p> <p><b>RETURNS:</b> zero if not end of file; non-zero if end of file</p>
<code>fileOpGetLastError</code>	<p>Get error code from last operation on stream, and clear the last error code value. Doesn't change status of EOF or I/O errors —use <a href="#"><u>FileClearerr</u></a> to reset all error codes.</p>
	<p><b>ARGUMENTS:</b> <code>stream</code> = open stream handle <code>valueP</code> = NULL <code>valueLenP</code> = NULL</p> <p><b>RETURNS:</b> Error code from last file stream operation</p>
<code>fileOpClearError</code>	<p>Clear I/O and EOF error status and last error.</p>
	<p><b>ARGUMENTS:</b> <code>stream</code> = open stream handle <code>valueP</code> = NULL <code>valueLenP</code> = NULL</p> <p><b>RETURNS:</b> zero on success; <code>fileErr...</code> on error</p>
<code>fileOpGetIOErrorStatus</code>	<p>Get I/O error status (like C runtime's <code>ferror</code>). Use <a href="#"><u>FileClearerr</u></a> to clear this error status.</p>
	<p><b>ARGUMENTS:</b> <code>stream</code> = open stream handle <code>valueP</code> = NULL <code>valueLenP</code> = NULL</p> <p><b>RETURNS:</b> zero if not I/O error; non-zero if I/O error is pending.</p>

<code>fileOpGetCreatedStatus</code>	<p>Find out whether file was created by <a href="#">FileOpen</a> function</p> <p><b>ARGUMENTS:</b> <code>stream</code> = open stream handle <code>valueP</code> = Pointer to Boolean <code>valueLenP</code> = Pointer to Int32 variable set to <code>sizeof(Boolean)</code></p> <p><b>RETURNS:</b> zero on success; <code>fileErr...</code> on error. The Boolean variable will be set to non-zero if the file was created.</p>
<code>fileOpGetOpenDbRef</code>	<p>Get the open database reference (handle) of the underlying database that implements the stream (NULL if none); this is needed for performing Palm OS-specific operations on the underlying database, such as changing or getting creator and type, version, backup/reset bits, and so on.</p> <p><b>ARGUMENTS:</b> <code>stream</code> = open stream handle <code>valueP</code> = Pointer to <code>DmOpenRef</code> variable <code>valueLenP</code> = Pointer to Int32 variable set to <code>sizeof(DmOpenRef)</code></p> <p><b>RETURNS:</b> zero on success; <code>fileErr...</code> on error. The <code>DmOpenRef</code> variable will be set to the file's open db reference that may be passed to Data Manager calls;</p> <p><b>WARNING:</b> Do not make any changes to the data of the underlying database -- doing so will corrupt the file stream.</p>
<code>fileOpFlush</code>	Flush any cached data to storage.

## File Streaming

### File Streaming Functions

---

ARGUMENTS:

stream = open stream handle  
valueP = NULL  
valueLenP = NULL

RETURNS:

zero on success; fileErr... on error;

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [FileClearerr](#), [FileEOF](#), [FileError](#), [FileFlush](#),  
[FileGetLastError](#), [FileRewind](#)

## FileDelete

**Purpose** Deletes the specified file stream from the specified card. Only a closed stream may be passed to this function.

**Declared In** FileStream.h

**Prototype** Err FileDelete (UInt16 cardNo, const Char \*nameP)

**Parameters** cardNo Card on which the file stream to delete resides. Currently, no Palm OS® devices support multiple cards, so this value must be 0.  
nameP String that is the name of the stream to delete.

**Result** 0 if no error, or a fileErr code if an error occurs. See the section “[File Streaming Error Codes](#)” for more information.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [FileOpen](#)

## FileDmRead

<b>Purpose</b>	A macro that reads data from a file stream into a chunk, record, or resource residing in a database.												
<b>Declared In</b>	FileStream.h												
<b>Prototype</b>	<pre>Int32 FileDmRead (FileHand stream,                    void *startOfDmChunkP, Int32 destOffset,                    Int32 objSize, Int32 numObj, Err *errP)</pre>												
<b>Parameters</b>	<table><tr><td>--&gt; stream</td><td>Handle to open stream.</td></tr><tr><td>--&gt; startOfDmChunkP</td><td>Pointer to beginning of chunk, record or resource residing in a database.</td></tr><tr><td>destOffset</td><td>Offset from startOfDmChunkP (base pointer) to the destination area (must be <math>\geq 0</math>).</td></tr><tr><td>objSize</td><td>Size of each stream object to read.</td></tr><tr><td>numObj</td><td>Number of stream objects to read.</td></tr><tr><td>&lt;-&gt; errP</td><td>Pointer to variable that is to hold the error code returned by this function. Pass NULL to ignore. See the section "<a href="#">File Streaming Error Codes</a>" for more information.</td></tr></table>	--> stream	Handle to open stream.	--> startOfDmChunkP	Pointer to beginning of chunk, record or resource residing in a database.	destOffset	Offset from startOfDmChunkP (base pointer) to the destination area (must be $\geq 0$ ).	objSize	Size of each stream object to read.	numObj	Number of stream objects to read.	<-> errP	Pointer to variable that is to hold the error code returned by this function. Pass NULL to ignore. See the section " <a href="#">File Streaming Error Codes</a> " for more information.
--> stream	Handle to open stream.												
--> startOfDmChunkP	Pointer to beginning of chunk, record or resource residing in a database.												
destOffset	Offset from startOfDmChunkP (base pointer) to the destination area (must be $\geq 0$ ).												
objSize	Size of each stream object to read.												
numObj	Number of stream objects to read.												
<-> errP	Pointer to variable that is to hold the error code returned by this function. Pass NULL to ignore. See the section " <a href="#">File Streaming Error Codes</a> " for more information.												
<b>Result</b>	The number of whole objects that were read—note that the number of objects actually read may be less than the number requested.												
<b>Comments</b>	When the number of objects actually read is less than the number requested, you may be able to determine the cause of this result by examining the return value of the errP parameter or by calling the <a href="#">FileGetLastError</a> function. If the cause is insufficient data in the stream to satisfy the full request, the current stream position is at end-of-file and the “end of file” indicator is set. If a non-NUL pointer was passed as the value of the errP parameter when the FileDmRead function was called and an error was encountered, *errP holds a non-zero error code when the function returns. In												

## File Streaming

### File Streaming Functions

---

In addition, the [FileError](#) and [FileEOF](#) functions may be used to check for I/O errors.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [FileRead](#), [FileError](#), [FileEOF](#)

## FileEOF

**Purpose** Get end-of-file status (err = fileErrEOF indicates end of file condition).

**Declared In** FileStream.h

**Prototype** Err FileEOF (FileHand stream)

**Parameters** --> stream Handle to open stream.

**Result** 0 if *not* end of file; non-zero if end of file. See the section “[File Streaming Error Codes](#)” for more information.

**Comments** This function’s behavior is similar to that of the feof function provided by the C programming language runtime library.  
Use [FileClearerr](#) to clear the I/O error status.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [FileClearerr](#), [FileGetLastError](#), [FileRewind](#)

## FileError

<b>Purpose</b>	Get I/O error status.
<b>Declared In</b>	FileStream.h
<b>Prototype</b>	Err FileError (FileHand stream)
<b>Parameters</b>	--> stream Handle to open stream.
<b>Result</b>	0 if no error, and non-zero if an I/O error indicator has been set for this stream. See the section “ <a href="#">File Streaming Error Codes</a> ” for more information.
<b>Comments</b>	This function’s behavior is similar to that of the C programming language’s <code>ferror</code> runtime function. Use <a href="#">FileClearerr</a> to clear the I/O error status.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">FileClearerr</a> , <a href="#">FileGetLastError</a> , <a href="#">FileRewind</a>

## FileFlush

<b>Purpose</b>	Flush cached data to storage.
<b>Declared In</b>	FileStream.h
<b>Prototype</b>	Err FileFlush (FileHand stream)
<b>Parameters</b>	--> stream Handle to open stream.
<b>Result</b>	0 if no error, or a <code>fileErr</code> code if an error occurs. See the section “ <a href="#">File Streaming Error Codes</a> ” for more information.
<b>Comments</b>	It is not always necessary to call this function explicitly—certain operations flush the contents of a stream automatically; for example,

## File Streaming

### *File Streaming Functions*

---

streams are flushed when they are closed. Because this function's behavior is similar to that of the `fflush` function provided by the C programming language runtime library, you only need to call it explicitly under circumstances similar to those in which you would call `fflush` explicitly.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

## FileGetLastError

**Purpose** Get error code from last operation on file stream, and clear the last error code value (will not change end of file or I/O error status -- use [`FileClearerr`](#) to reset all error codes)

**Declared In** `FileStream.h`

**Prototype** `Err FileGetLastError (FileHand stream)`

**Parameters** `--> stream` Handle to open stream.

**Result** Error code returned by the last file stream operation. See the section "[File Streaming Error Codes](#)" for more information.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [`FileClearerr`](#), [`FileEOF`](#), [`FileError`](#)

## FileOpen

**Purpose** Open existing file stream or create an open file stream for I/O in the mode specified by the `openMode` parameter.

**Declared In** `FileStream.h`

**Prototype**

```
FileHand FileOpen (UInt16 cardNo,
const Char *nameP, UInt32 type, UInt32 creator,
UInt32 openMode, Err *errP)
```

<b>Parameters</b>	<code>cardNo</code>	Card on which the file stream to open resides. Currently, no Palm Powered™ devices support multiple cards, so this value must be 0.
	<code>--&gt; nameP</code>	Pointer to text string that is the name of the file stream to open or create. This value must be a valid name—no wildcards allowed, must not be NULL.
	<code>type</code>	File type of stream to open or create. Pass 0 for wildcard, in which case <code>sysFileTFileStream</code> is used if the stream needs to be created and <code>fileModeTemporary</code> is not specified. If <code>type</code> is 0 and <code>fileModeTemporary</code> is specified, then <code>sysFileTTemp</code> is used for the file type of the stream this function creates.
	<code>creator</code>	Creator of stream to open or create. Pass 0 for wildcard, in which case the current application's creator ID is used for the creator of the stream this function creates.
	<code>openMode</code>	Mode in which to open the file stream. You must specify only one primary mode selector. Additionally, you can use the   operator (bitwise inclusive OR) to append one or more secondary mode selectors to the primary mode selector. See “ <a href="#">Primary Open Mode Constants</a> ” and “ <a href="#">Secondary Open Mode Constants</a> ” for the list of possible values.

## File Streaming

### File Streaming Functions

---

<--> errP

Pointer to variable that is to hold the error code returned by this function. Pass NULL to ignore. See the section “[File Streaming Error Codes](#)” for a list of error codes.

**Result** If successful, returns a handle to an open file stream; otherwise, returns 0.

In some cases, on some platforms, [FileOpen](#) returns a non-zero value when it has failed to open a file; thus, it is always a good idea to check the errP parameter value to determine if an error has occurred.

**Comments** The FileModeReadOnly, FileModeReadWrite, FileModeUpdate, and FileModeAppend modes are mutually exclusive—pass only one of them to the [FileOpen](#) function!

When the FileModeTemporary open mode is used and the file type passed to [FileOpen](#) is 0, the [FileOpen](#) function uses sysFileTTTemp (defined in SystemMgr.rh) for the file type, as recommended. In future versions of Palm OS, this configuration will enable the automatic cleanup of undeleted temporary files after a system crash. Automatic post-crash cleanup is not implemented in current versions of Palm OS.

To open a file stream even if it has a different type and creator than specified, pass the FileModeAnyTypeCreator selector as a flag in the openMode parameter to the [FileOpen](#) function.

The FileModeLeaveOpen mode is an esoteric option that most applications should not use. It may be useful for a library that needs to open a stream from the current application’s context and keep it open even after the current application quits. By default, Palm OS automatically closes all databases that were opened in a particular application’s context when that application quits. The FileModeLeaveOpen option overrides this default behavior.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

## FileRead

**Purpose** A macro that reads data from a stream into a buffer. Do not use this macro to read data into a chunk, record or resource residing in a database—you must use the [FileDmRead](#) macro for such operations.

**Declared In** FileStream.h

**Prototype** Int32 FileRead (FileHand stream, void \*bufP,  
Int32 objSize, Int32 numObj, Err \*errP)

**Parameters**

--> stream	Handle to open stream.
--> bufP	Pointer to beginning of buffer into which data is read
objSize	Size of each stream object to read.
numObj	Number of stream objects to read.
<-> errP	Pointer to variable that is to hold the error code returned by this function. Pass NULL to ignore. See the section " <a href="#">File Streaming Error Codes</a> " for a list of error codes.

**Result** The number of whole objects that were read—note that the number of objects actually read may be less than the number requested.

**Comments** Do not use this macro to read data into a chunk, record or resource residing in a database—you must use the [FileDmRead](#) macro for such operations.

When the number of objects actually read is fewer than the number requested, you may be able to determine the cause of this result by examining the return value of the `errP` parameter or by calling the [FileGetLastError](#) function. If the cause is insufficient data in the stream to satisfy the full request, the current stream position is at end-of-file and the “end of file” indicator is set. If a non-NULL pointer was passed as the value of the `errP` parameter when the `FileRead` function was called and an error was encountered, `*errP` holds a non-zero error code when the function returns. In

## File Streaming

### *File Streaming Functions*

---

In addition, the [FileError](#) and [FileEOF](#) functions may be used to check for I/O errors.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [FileDmRead](#)

## FileRewind

**Purpose** Reset position marker to beginning of stream and clear all error codes.

**Declared In** FileStream.h

**Prototype** Err FileRewind (FileHand stream)

**Parameters** --> stream Handle to open stream.

**Result** 0 if no error, or a fileErr code if an error occurs. See the section “[File Streaming Error Codes](#)” for more information.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [FileSeek](#), [FileTell](#), [FileClearerr](#), [FileEOF](#), [FileError](#), [FileGetLastError](#)

## FileSeek

**Purpose** Set current position within a file stream, extending the stream as necessary if it was opened with write access.

**Declared In** FileStream.h

**Prototype** Err FileSeek (FileHand stream, Int32 offset, FileOriginEnum origin)

**Parameters** --> stream Handle to open stream.

offset	Position to set, expressed as the number of bytes from origin. This value may be positive, negative, or 0.
origin	Describes the origin of the position change. Possible values are:  fileOriginBeginning From the beginning (first data byte of file).  fileOriginCurrent From the current position.  fileOriginEnd From the end of file (one position beyond last data byte).

**Result** 0 if no error, or a `fileErr` code if an error occurs. See the section “[File Streaming Error Codes](#)” for more information.

**Comments** Attempting to seek beyond end-of-file in a read-only stream results in an I/O error.

This function’s behavior is similar to that of the `fseek` function provided by the C programming language runtime library.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [FileRewind](#), [FileTell](#)

## FileTell

**Purpose** Retrieves the current position and, optionally, file size, of a stream.

**Declared In** `FileStream.h`

**Prototype** `Int32 FileTell (FileHand stream, Int32 *fileSizeP, Err *errP)`

**Parameters** `--> stream` Handle to open stream.

## File Streaming

### File Streaming Functions

---

<-> fileSizeP	Pointer to variable that holds value describing size of stream in bytes when this function returns. Pass NULL to ignore.
<-> errP	Pointer to variable that is to hold the error code returned by this function. Pass NULL to ignore. See the section " <a href="#">File Streaming Error Codes</a> " for a list of possible error codes.

**Result** If successful, returns current position, expressed as an offset in bytes from the beginning of the stream. If an error was encountered, returns -1 as a signed long integer.

**Comments** The FileTell function can return the size of the input stream; as such, it provides some of the functionality of the standard C library stat function. Note, however, that unlike the stat function, FileTell requires that the file be open.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [FileRewind](#), [FileSeek](#)

## FileTruncate

**Purpose** Truncate the file stream to a specified size; not allowed on streams open in destructive read mode or read-only mode.

**Declared In** FileStream.h

**Prototype** Err FileTruncate (FileHand stream, Int32 newSize)

**Parameters** --> stream Handle of open stream.  
newSize New size; must not exceed current stream size.

**Result** 0 if no error, or a fileErr code if an error occurs. See the section "[File Streaming Error Codes](#)" for a list of possible error codes.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [FileTell](#)

## FileWrite

**Purpose** Write data to a stream.

**Declared In** FileStream.h

**Prototype** Int32 FileWrite (FileHand stream,  
const void \*dataP, Int32 objSize, Int32 numObj,  
Err \*errP)

**Parameters**

--> stream	Handle to open stream.
--> dataP	Pointer to buffer holding data to write.
objSize	Size of each stream object to write; must be $\geq 0$ .
numObj	Number of stream objects to write.
<-> errP	Optional pointer to variable that holds the error code returned by this function. Pass NULL to ignore. See the section " <a href="#">File Streaming Error Codes</a> " for a list of possible error codes.

**Result** The number of whole objects that were written—note that the number of objects actually written may be less than the number requested. Should available storage be insufficient to satisfy the entire request, as much of the requested data as possible is written to the stream, which may result in the last object in the stream being incomplete.

**Comments** Writing to files opened without write access or those that are in destructive read state is not allowed; thus, you cannot call the [FileWrite](#), [FileSeek](#), or [FileTruncate](#) functions on a stream that is in destructive read mode. One exception to this rule applies to streams that were opened in “write + append” mode and then switched into destructive read state. In this case, the [FileWrite](#) function can append data to the stream, but it also preserves the

## **File Streaming**

### *File Streaming Functions*

---

current stream position so that subsequent reads pick up where they left off (you can think of this as a pseudo-pipe).

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

## File Streaming Error Codes

This section lists all error codes returned by the file streaming functions.

Error Code	Value	Meaning
fileErrMemErr	(fileErrorClass   1)	Out of memory error
fileErrInvalidParam	(fileErrorClass   2)	Invalid parameter value passed
fileErrCorruptFile	(fileErrorClass   3)	Alleged stream is corrupted, invalid, or not a stream
fileErrNotFound	(fileErrorClass   4)	Couldn't find the stream
fileErrTypeCreatorMismatch	(fileErrorClass   5)	Type and/or creator not what was specified
fileErrReplaceError	(fileErrorClass   6)	Couldn't replace existing stream
fileErrCreateError	(fileErrorClass   7)	Couldn't create new stream
fileErrOpenError	(fileErrorClass   8)	Generic open error
fileErrInUse	(fileErrorClass   9)	Stream couldn't be opened or deleted because it is in use
fileErrReadOnly	(fileErrorClass   10)	Couldn't open in write mode because existing stream is read-only
fileErrInvalidDescriptor	(fileErrorClass   11)	Invalid file descriptor (FileHandle)

## File Streaming

### *File Streaming Error Codes*

---

Error Code	Value	Meaning
fileErrCloseError	(fileErrorClass   12)	Error closing the stream
fileErrOutOfBounds	(fileErrorClass   13)	Attempted operation went out of bounds of the stream
fileErrPermissionDenied	(fileErrorClass   14)	Couldn't write to a stream open for read-only access
fileErrIOError	(fileErrorClass   15)	Generic I/O error
fileErrEOF	(fileErrorClass   16)	End-of-File error
fileErrNotStream	(fileErrorClass   17)	Attempted to open an entity that is not a stream

---

# Float Manager

---

This chapter provides reference material for the Float Manager API as follows:

- [Float Manager Data Structures](#)
- [Float Manager Functions](#)

The Float Manager API is declared in the header file `FloatMgr.h`. For more information on the Float Manager, see the section “[Floating-Point](#)” in the *Palm OS Programmer’s Companion*, vol. I.

## Float Manager Data Structures

### **FlpCompDouble**

Float Manager functions accept and require values of type `FlpDouble`. The `FlpCompDouble` union allows you to declare values that can be interpreted either as a double or as an `FlpDouble`. As well, this union contains fields that provide easy access to the component parts of the double-precision floating-point number.

```
typedef union {
    double d;
    FlpDouble fd;
    UInt32 ul[2];
    FlpDoubleBits fdb;
} FlpCompDouble
```

## Float Manager

### Float Manager Data Structures

---

#### Field Descriptions

d	Provides access to the value as a double.
fd	Provides access to the value as a FlpDouble, which can be passed to or received from many Float Manager functions.
ul	Provides access to the value as two long integers.
fdb	Provides access to specific fields.

#### FlpDoubleBits

This structure provides direct access to the component parts of an IEEE-754 double-precision floating-point number. Use the [FlpCompDouble](#) union to convert numbers of type double to and from FlpDoubleBits.

```
typedef struct {
    UInt32 sign : 1;
    Int32 exp : 11;
    UInt32 manH : 20;
    UInt32 manL;
} FlpDoubleBits
```

#### Field Descriptions

sign	The sign bit. You can also use the <a href="#">FlpGetSign</a> macro to obtain the sign bit, and the <a href="#">FlpNegate</a> , <a href="#">FlpSetNegative</a> , and <a href="#">FlpSetPositive</a> macros to set the sign bit.
exp	The bits that make up the exponent. You can also use the <a href="#">FlpGetExponent</a> macro to obtain the exponent value.
manH	The most-significant 20 bits of the mantissa.
manL	The least-significant 32 bits of the mantissa.

# Float Manager Functions

## FlpAToF

**Purpose** Convert a null-terminated ASCII string to a 64-bit floating-point number. The string must have the format:

[+ | -] [digits] [.] [digits] [e | E [+ | -] [digits]]

**Declared In** `FloatMgr.h`

**Prototype** `FlpDouble FlpAToF (const Char *s)`

**Parameters** `-> s` Pointer to the string to be converted.

**Result** Returns the value of the string as a floating-point number.

**Comment** The mantissa of the number is limited to 32 bits.

This function is close to being compatible with the ISO C library function `atof`. `atof` requires the form:

[+ | -] digits [.] [digits] [(e | E) [+ | -] digits]

In order to maintain backward compatibility with the Float Manager in Palm OS 1.0 (which could be used up to, but not including, Palm OS 4.0), this function considers all of the “digits” sections to be optional. Here’s a table showing the ISO and Palm OS behavior with some sample strings:

String	ISO	>= Palm OS 4.0	< Palm OS 4.0 <sup>1</sup>	Notes
“+”	+0	+0	+0	
“.3”	0.3	0.3	0.3	
“0.3e123”	0.3e123	0.3e123	0.3e12	The old Float Manager only allowed a 1 or 2 digit exponent.

## Float Manager

### Float Manager Functions

---

String	ISO	>= Palm OS 4.0	< Palm OS 4.0 <sup>1</sup>	Notes
“+1”	1	1	+0	The old Float Manager doesn't allow a leading '+' sign.
“1e+2”	1e2	1e2	1	The old Float Manager doesn't allow a '+' sign in the exponent.
“0.3E3”	0.3e3	0.3e3	0.3	The old Float Manager doesn't allow a capital 'E' to mark the exponent.
“4294967297”	4294967297	4294967297	1	The old Float Manager uses an unsigned long and wraps around.

1. Using the old Float Manager documented in [Appendix C, “1.0 Float Manager.”](#) on page 2355.

Unlike `atof`, `FlpAToF` doesn't accept leading white-space characters and it doesn't accept decimal point characters other than ‘.’.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present. GCC users must use [`FlpBufferAToF`](#) instead of this function.

**See Also** [`FlpFToA`](#)

## FlpBase10Info

**Purpose** Extract detailed information on the base 10 form of a floating-point number: the base 10 mantissa, exponent, and sign.

**Declared In** `FloatMgr.h`

**Prototype** `Err FlpBase10Info (FlpDouble a, UInt32 *mantissaP, Int16 *exponentP, Int16 *signP)`

**Parameters** `-> a` The floating-point number.

<- mantissaP      The base 10 mantissa.  
<- exponentP      The base 10 exponent.  
<- signP            The sign: 1 if the number is negative, 0 otherwise.

**Result**      Returns 0 if no error, or `f1pErrOutOfRange` if the supplied floating-point number is either not a number (NaN) or is infinite.

**Comments**      The mantissa is normalized so it contains at least 8 significant digits when printed as an integer value.

**Compatibility**      Implemented only if [2.0 New Feature Set](#) is present.

**See Also**      [F1pGetExponent](#), [F1pGetSign](#)

## F1pBufferAToF

**Purpose**      Convert a null-terminated ASCII string to a floating-point number. The string must be in the format: [-]x[.]yyyyyyyy [e[-]zz]

**Declared In**      `FloatMgr.h`

**Prototype**      `void F1pBufferAToF (FlpDouble *result,  
                      const Char *s)`

**Parameters**      <- result      Pointer to the structure into which the return value is placed.  
                      -> s          Pointer to the null-terminated ASCII string to be converted.

**Result**      Returns the value of the string as a floating-point number.

**Comments**      See [F1pATO](#)F for a complete description of this function.

**Compatibility**      Implemented only if [2.0 New Feature Set](#) is present. Because the Palm OS ABI was not well-specified in this area, GCC by default implemented structure return differently from the compiler used to

## Float Manager

### Float Manager Functions

---

build the ROM. As a result, GCC users must use this function instead of [FlpAToF](#). CodeWarrior users can use either function; they are binary compatible.

## FlpBufferCorrectedAdd

**Purpose** Adds two floating-point numbers and corrects for least-significant-bit errors when the result should be zero but is instead very close to zero.

**Declared In** `FloatMgr.h`

**Prototype** `void FlpBufferCorrectedAdd (FlpDouble *result,  
FlpDouble firstOperand, FlpDouble secondOperand,  
Int16 howAccurate)`

**Parameters** `<- result` Pointer to the structure into which the return value is placed.

`-> firstOperand` The first of the two numbers to be added.

`-> secondOperand` The second of the two numbers to be added.

`-> howAccurate` The smallest difference in exponents that won't force the result to zero. The value returned from this function is forced to zero if the difference between exponents in the smaller of the two operands and the result exceeds this value. Supply a value of zero for this parameter to obtain the default level of accuracy (which is equivalent to a `howAccurate` value of 48).

**Result** Returns the calculated result.

**Comments** See [FlpCorrectedAdd](#) for a complete description of this function.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present. Because the Palm OS ABI was not well-specified in this area, GCC by default

implemented structure return differently from the compiler used to build the ROM. As a result, GCC users must use this function instead of [FlpCorrectedAdd](#). CodeWarrior users can use either function; they are binary compatible.

## FlpBufferCorrectedSub

**Purpose** Subtracts two floating-point numbers and corrects for least-significant-bit errors when the result should be zero but is instead very close to zero.

**Declared In** `FloatMgr.h`

**Prototype** `void FlpBufferCorrectedSub (FlpDouble *result,  
FlpDouble firstOperand, FlpDouble secondOperand,  
Int16 howAccurate)`

**Parameters**

<- <code>result</code>	Pointer to the structure into which the return value is placed.
-> <code>firstOperand</code>	The value from which <code>secondOperand</code> is to be subtracted.
-> <code>secondOperand</code>	The value to subtract from <code>firstOperand</code> .
-> <code>howAccurate</code>	The smallest difference in exponents that won't force the result to zero. The value returned from this function is forced to zero if the difference between exponents in the smaller of the two operands and the result exceeds this value. Supply a value of zero for this parameter to obtain the default level of accuracy (which is equivalent to a <code>howAccurate</code> value of 48).

**Result** Returns the calculated result.

**Comments** See [FlpCorrectedSub](#) for a complete description of this function.

## Float Manager

### Float Manager Functions

---

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present. Because the Palm OS ABI was not well-specified in this area, GCC by default implemented structure return differently from the compiler used to build the ROM. As a result, GCC users must use this function instead of [FlpCorrectedSub](#). CodeWarrior users can use either function; they are binary compatible.

## FlpCorrectedAdd

**Purpose** Adds two floating-point numbers and corrects for least-significant-bit errors when the result should be zero but is instead very close to zero.

**Declared In** `FloatMgr.h`

**Prototype** `FlpDouble FlpCorrectedAdd  
(FlpDouble firstOperand, FlpDouble secondOperand,  
Int16 howAccurate)`

**Parameters**

- > `firstOperand`  
The first of the two numbers to be added.
- > `secondOperand`  
The second of the two numbers to be added.
- > `howAccurate` The smallest difference in exponents that won't force the result to zero. The value returned from `FlpCorrectedAdd` is forced to zero if, when the exponent of the result of the addition is subtracted from the exponent of the smaller of the two operands, the difference exceeds the value specified for `howAccurate`. Supply a value of zero for this parameter to obtain the default level of accuracy (which is equivalent to a `howAccurate` value of 48).

**Result** Returns the calculated result.

**Comments** Adding or subtracting a large number and a small number produces a result similar in magnitude to the larger number. Adding or

subtracting two numbers that are similar in magnitude can, depending on their signs, produce a result with a very small exponent (that is, a negative exponent that is large in magnitude). If the difference between the result's exponent and that of the operands is close to the number of significant bits expressible by the mantissa, it is quite possible that the result should in fact be zero.

There also exist cases where it may be useful to retain accuracy in the low-order bits of the mantissa. For instance:  $99999999 + 0.00000001 - 99999999$ . However, unless the fractional part is an exact (negative) power of two, it is doubtful that what few bits of mantissa that are available will be enough to properly represent the fractional value. In this example, the 99999999 requires 26 bits, leaving 26 bits for the .00000001; this guarantees inaccuracy after the subtraction.

The problem arises from the difficulty in representing decimal fractions such as 0.1 in binary. After about three successive additions or subtractions, errors begin to appear in the least significant bits of the mantissa. If the value represented by the most significant bits of the mantissa is then subtracted away, the least significant bit error is normalized and becomes the actual result—when in fact the result should be zero.

This problem is only an issue for addition and subtraction.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present. GCC users must use [F1pBufferCorrectedAdd](#) instead of this function.

**See Also** [F1pCorrectedSub](#)

## Float Manager

### *Float Manager Functions*

---

## FlpCorrectedSub

**Purpose** Subtracts two floating-point numbers and corrects for least-significant-bit errors when the result should be zero but is instead very close to zero.

**Declared In** `FloatMgr.h`

**Prototype** `FlpDouble FlpCorrectedSub  
(FlpDouble firstOperand, FlpDouble secondOperand,  
Int16 howAccurate)`

**Parameters**

-> <code>firstOperand</code>	The value from which <code>secondOperand</code> is to be subtracted.
-> <code>secondOperand</code>	The value to subtract from <code>firstOperand</code> .
-> <code>howAccurate</code>	The smallest difference in exponents that won't force the result to zero. The value returned from <code>FlpCorrectedSub</code> is forced to zero if, when the exponent of the result of the subtraction is subtracted from the exponent of the smaller of the two operands, the difference exceeds the value specified for <code>howAccurate</code> . Supply a value of zero for this parameter to obtain the default level of accuracy (which is equivalent to a <code>howAccurate</code> value of 48).

**Result** Returns the calculated result.

**Comments** See the comments for [FlpCorrectedAdd](#).

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present. GCC users must use [FlpBufferCorrectedSub](#) instead of this function.

## FlpFToA

<b>Purpose</b>	Convert a floating-point number to a null-terminated ASCII string in exponential format: [-] x.yyyyyyye [-] zz
<b>Declared In</b>	FloatMgr.h
<b>Prototype</b>	Err FlpFToA (FlpDouble a, Char *s)
<b>Parameters</b>	-> a Floating-point number. <- s Pointer to buffer to contain the ASCII string.
<b>Result</b>	Returns 0 if no error, or flpErrOutOfRange if the supplied value is infinite or is not a number. In this case, the buffer is set to the string "INF", "-INF", or "NaN" as appropriate.
<b>Compatibility</b>	Implemented only if <a href="#">2.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">FlpAToF</a>

## FlpGetExponent

<b>Purpose</b>	Macro that returns the exponent of a 64-bit floating-point value. The returned value has the bias applied, so it ranges from -1023 to +1024.
<b>Declared In</b>	FloatMgr.h
<b>Prototype</b>	FlpGetExponent (x)
<b>Parameters</b>	-> x The value from which the exponent is to be extracted.
<b>Result</b>	Returns a UInt32 containing the exponent of the specified value.
<b>Compatibility</b>	Implemented only if <a href="#">2.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">FlpBase10Info</a> , <a href="#">FlpGetSign</a>

## Float Manager

### *Float Manager Functions*

---

## FlpGetSign

**Purpose** Macro that returns the sign of a 64-bit floating-point value.

**Declared In** `FloatMgr.h`

**Prototype** `FlpGetSign (x)`

**Parameters** `-> x` The value from which the sign bit is to be extracted.

**Result** Returns a `UInt32` with a nonzero value if the specified value is negative, and with a zero value if it is positive.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

**See Also** [FlpBase10Info](#), [FlpGetExponent](#), [FlpNegate](#),  
[FlpSetNegative](#), [FlpSetPositive](#)

## FlpIsZero

**Purpose** Macro that returns whether the specified 64-bit floating-point value is zero.

**Declared In** `FloatMgr.h`

**Prototype** `FlpIsZero (x)`

**Parameters** `-> x` The value for which the sign bit is desired.

**Result** Returns a `UInt32` with a nonzero value if the specified value is zero, and with a zero value if the specified value is other than zero.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## FlpNegate

<b>Purpose</b>	Macro that changes the sign bit of a 64-bit floating-point number.
<b>Declared In</b>	FloatMgr.h
<b>Prototype</b>	FlpNegate (x)
<b>Parameters</b>	- > x The value in which the sign bit is to be changed.
<b>Result</b>	Returns a 64-bit floating-point value which is the negative of the value specified by x.
<b>Compatibility</b>	Implemented only if <a href="#">2.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">FlpGetSign</a> , <a href="#">FlpSetNegative</a> , <a href="#">FlpSetPositive</a>

## FlpSetNegative

<b>Purpose</b>	Macro that ensures that a 64-bit floating-point number is negative.
<b>Declared In</b>	FloatMgr.h
<b>Prototype</b>	FlpSetNegative (x)
<b>Parameters</b>	- > x The value that is to be forced negative.
<b>Result</b>	If the supplied 64-bit floating-point value is negative, that value is returned unchanged. If the supplied value is positive, the negative of that value is returned.
<b>Compatibility</b>	Implemented only if <a href="#">2.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">FlpGetSign</a> , <a href="#">FlpNegate</a> , <a href="#">FlpSetPositive</a>

## **Float Manager**

### *Float Manager Functions*

---

## **FlpSetPositive**

**Purpose** Macro that ensures that a 64-bit floating-point number is positive.

**Declared In** `FloatMgr.h`

**Prototype** `FlpSetPositive (x)`

**Parameters** `-> x` The value that is to be forced positive.

**Result** If the supplied 64-bit floating-point value is positive, that value is returned unchanged. If the supplied value is negative, its absolute value is returned.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

**See Also** [FlpGetSign](#), [FlpNegate](#), [FlpSetNegative](#)

## **FlpVersion**

**Purpose** Returns the version number of the Float Manager.

**Declared In** `FloatMgr.h`

**Prototype** `UInt32 FlpVersion (void)`

**Parameters** None.

**Result** Returns the version number of the Float Manager. The current version is represented by the constant `flpVersion`, which is defined in `FloatMgr.h`.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

# Fonts

---

This chapter provides the following information regarding font support:

- [Font Data Structures](#)
- [Font Constants](#)
- [Font Resources](#)
- [Font Functions](#)

The header files `Font.h` and `FontSelect.h` declare the API that this chapter describes. For more information on fonts, see [Chapter 8, “Text,”](#) on page 251 of the *Palm OS Programmer’s Companion*, vol. I.

## Font Data Structures

### FontCharInfoPtr

The `FontCharInfoPtr` type points to a [FontCharInfoType](#) structure.

```
typedef FontCharInfoType *FontCharInfoPtr;
```

### FontCharInfoType

The `FontCharInfoType` structure defines an entry in the offset/width table for a font.

---

**WARNING!** Palm, Inc. does not support or provide backward compatibility for the `FontCharInfoType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

## Fonts

### Font Data Structures

---

```
typedef struct FontCharInfoTag {  
    Int8 offset;  
    Int8 width;  
} FontCharInfoType;
```

#### Field Descriptions

**offset** This value is not currently used and must be set to 0.

**width** The exact width in pixels of the glyph. You can retrieve this information using the function [FntWCharWidth](#) or [FntCharWidth](#).



## FontDensityType

The FontDensityType structure defines an entry in the densities array in the [FontTypeV2](#) structure. The densities array specifies the location of each set of glyphs within an extended font resource.

---

**WARNING!** Palm, Inc. does not support or provide backward compatibility for the FontDensityType structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct FontDensityTag {  
    Int16 density;  
    UInt32 glyphBitsOffset;  
} FontDensityType;
```

#### Field Description

**density** Either [kDensityLow](#) or [kDensityDouble](#).

**glyphBitsOffset** Offset in bytes from the beginning of the font data to the start of the font image for this density.

## Compatibility

This structure is only defined if the [High-Density Display Feature Set](#) is present.

## FontID

The FontID enum specifies the IDs of available fonts. A font can either be a system-defined font or an application-defined font. You can obtain the ID of the current font using [FntGetFont](#) and change the font using [FntSetFont](#).

```
enum fontID {
    stdFont = 0x00,
    boldFont,
    largeFont,
    symbolFont,
    symbol11Font,
    symbol17Font,
    ledFont,
    largeBoldFont,
    fntAppCustomBase = 0x80
};
typedef enum fontID FontID;
```

## Value Descriptions

stdFont	A small standard font used to display user input. This font is small to display as much text as possible.
boldFont	Same size as stdFont but bold for easier reading. Used for text labels in the user interface.
largeFont	A larger font provided as an alternative for users who find the standard font too small to read.
symbolFont	Contains many special characters such as arrows, Graffiti® Shift Indicators, and so on.

## Fonts

### Font Data Structures

---

symbol11Font	Contains the check boxes, the large left arrow, and the large right arrow.
symbol17Font	Contains the up and down arrows used for the repeating button scroll arrows and the dimmed version of the same arrows.
ledFont	Contains the numbers 0 through 9, −, ., and the comma (,). Used by the Calculator application for its numeric display.
largeBoldFont	In Palm OS® 3.0 and later only. Same size as largeFont but bold.
fntAppCustomBase	The first available ID for application-defined fonts.

## FontPtr

The FontPtr type defines a pointer to a [FontType](#) structure.

```
typedef FontType *FontPtr;
```

## FontType

The FontType structure defines a font resource's header. The fields in this structure give general information about the font. Following the structure are several tables that Palm OS uses to draw the font on the screen. See “[Font Resource](#)” on page 718 for more information about the font resource.

---

**WARNING!** Palm, Inc. does not support or provide backward compatibility for the `FontType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct FontTag {
    Int16 fontType;
    Int16 firstChar;
    Int16 lastChar;
    Int16 maxWidth;
```

```
Int16 kernMax;
Int16 nDescent;
Int16 fRectWidth;
Int16 fRectHeight;
Int16 owTLoc;
Int16 ascent;
Int16 descent;
Int16 leading;
Int16 rowWords;
} FontType;
```

### Field Descriptions

<code>fontType</code>	A mask providing the general characteristics of the font. When creating an application-defined font resource, use 0x9000.
<code>firstChar</code>	Character code of first glyph in the font.
<code>lastChar</code>	Character code of last glyph in the font.
<code>maxWidth</code>	The maximum width in pixels of any glyph. In Palm OS, there is currently no difference between this field and <code>fRectWidth</code> .
<code>kernMax</code>	This value is not currently used and must be set to 0.
<code>nDescent</code>	This value is not currently used and must be set to 0.
<code>fRectWidth</code>	A metric of the font image. In Palm OS, this metric is equivalent to the maximum width in pixels of any glyph in the font. Use <a href="#"><u>FntAverageCharWidth</u></a> to obtain this value.
<code>fRectHeight</code>	The height, including ascenders and descenders, of the glyphs in this font. Use <a href="#"><u>FntCharHeight</u></a> to obtain this value.

## Fonts

### Font Data Structures

---

owTLoc	The offset in 16-bit words from this field to the first byte of the offset/width table. The offset/width table is a table of <a href="#">FontCharInfoType</a> structures giving the width of each character in the font. Do not access the offset/width table directly. Use <a href="#">FntWCharWidth</a> or <a href="#">FntCharWidth</a> instead.
ascent	The distance in pixels from the top of the font rectangle to its baseline. Use <a href="#">FntBaseLine</a> to obtain this value.
descent	The distance in pixels from the baseline to the bottom of the font rectangle. Use <a href="#">FntDescenderHeight</a> to obtain this value.
leading	The font's leading, which is the vertical space between lines of text, in pixels. This field is unused in Palm OS and must be set to 0. If your font requires a leading value, add blank space to the bottom of each of your glyphs. The <a href="#">FntLineHeight</a> function returns the size of the font's character cell plus the leading.
rowWords	The number of 16-bit words stored for each row of a glyph's bitmap where fRectHeight is the number of rows.



## FontTypeV2

---

The FontTypeV2 structure defines the header for an extended font resource, which contains a separate set of glyphs for each screen density. Currently the only supported densities are kDensityLow and kDensityDouble. See “[Extended Font Resource](#)” on page 721 for more information.

**WARNING!** Palm, Inc. does not support or provide backward compatibility for the `FontCharTypeV2` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct FontTagV2 {  
    Int16 fontType;  
    Int16 firstChar;  
    Int16 lastChar;  
    Int16 maxWidth;  
    Int16 kernMax;  
    Int16 nDescent;  
    Int16 fRectWidth;  
    Int16 fRectHeight;  
    Int16 owTLoc;  
    Int16 ascent;  
    Int16 descent;  
    Int16 leading;  
    Int16 rowWords;  
    Int16 version;  
    Int16 densityCount;  
    FontDensityType densities[0];  
} FontTypeV2;
```

---

**NOTE:** All pixel values given in the fields below are in terms of the single-density font data.

---

### Field Descriptions

<code>fontType</code>	A mask providing the general characteristics of the font. When creating an application-defined extended font resource, use the value <code>fntExtendedFormatMask   0x9000</code> .
<code>firstChar</code>	Character code of first glyph in the font.
<code>lastChar</code>	Character code of last glyph in the font.

## Fonts

### Font Data Structures

---

maxWidth	The maximum width in pixels of any glyph. In Palm OS, there is currently no difference between this field and fRectWidth.
kernMax	This value is not currently used and must be set to 0.
nDescent	This value is not currently used and must be set to 0.
fRectWidth	A metric of the font image. In Palm OS, this metric is equivalent to the maximum width in pixels of any glyph in the font. Use <a href="#">FntAverageCharWidth</a> to obtain this value.
fRectHeight	The height, including ascenders and descenders, of the glyphs in this font. Use <a href="#">FntCharHeight</a> to obtain this value.
owTLoc	The offset in 16-bit words from this field to the first byte of the offset/width table. The offset/width table is a table of <a href="#">FontCharInfoType</a> structures giving the width of each character in the font. Do not access the offset/width table directly. Use <a href="#">FntWCharWidth</a> or <a href="#">FntCharWidth</a> instead.
ascent	The distance in pixels from the top of the font rectangle to its baseline. Use <a href="#">FntBaseLine</a> to obtain this value.
descent	The distance in pixels from the baseline to the bottom of the font rectangle. Use <a href="#">FntDescenderHeight</a> to obtain this value.
leading	The font's leading, which is the vertical space between lines of text, in pixels. This field is unused in Palm OS and must be set to 0. If your font requires a leading value, add blank space to the bottom of each of your glyphs. The <a href="#">FntLineHeight</a> function returns the size of the font's character cell plus the leading.

---

<code>rowWords</code>	The number of 16-bit words stored for each row of a glyph's bitmap where <code>fRectHeight</code> is the number of rows.
<code>version</code>	The version of the extended font resource. This value should be set to 1.
<code>densityCount</code>	The number of entries in the <code>densities</code> array.
<code>densities</code>	An array of one or more <a href="#">FontDensityType</a> structures identifying the glyphs for each supported density.

### Compatibility

This structure is only defined if the [High-Density Display Feature Set](#) is present.

## Font Constants

---

Constant	Value	Description
<code>checkboxFont</code>	<code>symbol11Font</code>	A convenience constant that points to the font containing the checkbox bitmap.
<code>fntMissingChar</code>	-1	The value used for a character that does not have a definition in the current font. The missing character symbol is usually an open rectangle.
<code>fntExtendedFormatMask</code>	0x0200	A constant used for the <code>fontType</code> field of a font to indicate that it is an extended font resource.
<code>fntTabChrWidth</code>	20	The width of the tab character in pixels.

---

## Fonts

### Font Resources

---

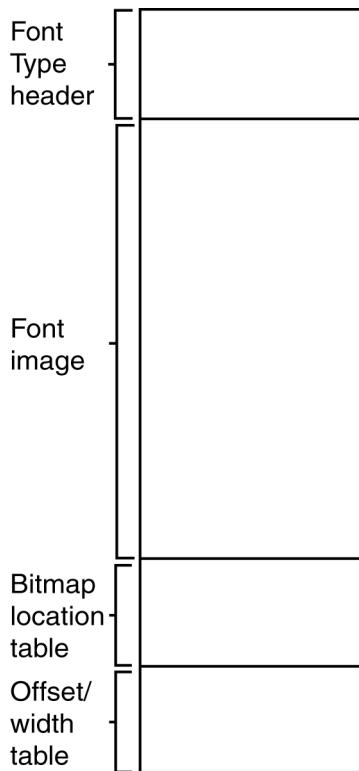
# Font Resources

## Font Resource

The font resource ('NFNT') represents a version 1 single-density font. This resource is the same as the Macintosh 'NFNT' resource with some restrictions. It contains a header followed by several tables that provide information about each glyph in the font.

[Figure 33.1](#) shows how the font resource is laid out in memory. [Table 33.1](#) describes each table within the font resource.

**Figure 33.1 Font resource ('NFNT')**



**Table 33.1 Font resource description**

Field	Description
FontType header	Contains general information about the glyphs in the font. See <a href="#">FontType</a> .
Font image	A raw bitmap image containing the packed character glyphs from left to right (see <a href="#">Figure 33.2</a> on page 720). This part of the resource tells Palm OS how to draw each character in the font. The height of the image is fRectHeight and the size is rowWords * 2 * fRectHeight.
Bitmap location table	Place glyphs sequentially in order of increasing character code. Leave at least a one-pixel wide vertical column of space to the right of each image so that there is space between characters when Palm OS draws text on the screen. If your font requires leading, leave horizontal space at the bottom of the characters as well. The font image must end with the glyph for the missing character symbol.
	A table of 16-bit words that specify the location of each glyph's entry in the font image. The location is specified as the bit offset from the start of the image to the glyph in the first row of the font image. The last entry in the table contains the offset of the column after the last bitmap. (See <a href="#">Figure 33.2</a> on page 720.)
	If you have skipped characters within an encoding, for each glyph that is missing, specify the same value for its location as the entry for the next glyph in the table.

## Fonts

### Font Resources

---

**Table 33.1 Font resource description (*continued*)**

Field	Description
Offset/width table	A table that specifies how wide each glyph in the font is. On Macintosh systems, this table also specifies how each glyph kerns. Palm OS does not support kerning, as the offset value is ignored.  Each entry in the offset/width table is two bytes long. The first byte should be 0, and the second byte should contain the glyph width, which must be greater than or equal to 0. If the glyph at this index does not have a bitmap in the font image, the values should be -1 and -1.

[Figure 33.2](#) shows an example of the font image for a font that defines glyphs for four characters (A, B, C, and the missing character symbol) and the portion of the bitmap location table that provides the offsets for these characters. The last entry in the bitmap location table is the offset to the column after the last bitmap, or 0x0014.

**Figure 33.2 Font image and bitmap location table**

Font image:	-----	-----	-----	-----	
	- ## -	# ## -	# # -	##### #	
	# - - # -	# - - # -	# - - # -	# - - #	
	# - - # -	# ## -	# - - -	# - - -	
	# ## -	# - - # -	# - - -	# - - -	
	# - - # -	# - - # -	# - - -	# - - -	
	# - - # -	# ## -	# # -	##### #	
Bitmap location table values:	0x0000	0x0005	0x000A	0x000F	0x0014



## New **Extended Font Resource**

The extended font resource ('nfnt') defines a font that supports multiple screen densities. Currently, only two screen densities are supported: the standard density of 80 dpi, as occurs on most devices that use a 160 X 160 pixel display, and double density of 160 dpi, as occurs on most devices that use a 320 X 320 pixel display. As shown in [Figure 33.3](#), the extended font resource is essentially:

- A [FontTypeV2](#) header giving all general information about the glyphs in the font. All metrics are in terms of the low-density version of the font.
- Tables for the low-density font. See “[Font Resource](#)” on page 718 for a description of these tables.
- The font image (set of glyphs) for each density specified by the font.

### **Compatibility**

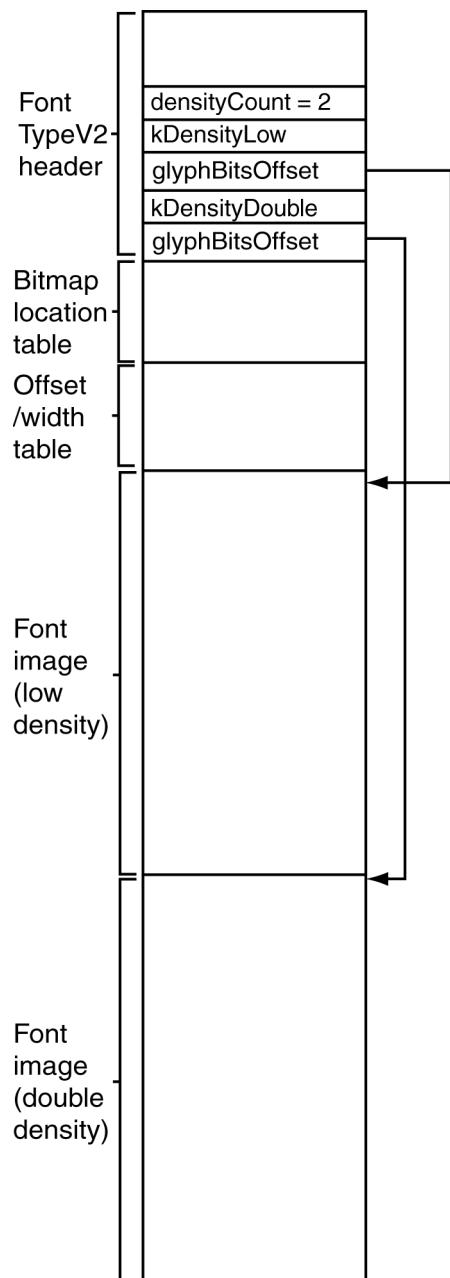
This resource is only defined if the [High-Density Display Feature Set](#) is present.

## Fonts

### Font Resources

---

**Figure 33.3 Extended font resource**



## Font Functions

### FntAverageCharWidth

**Purpose** Gets the *maximum* character width in pixels of the current font.

**Declared In** `Font.h`

**Prototype** `Int16 FntAverageCharWidth (void)`

**Parameters** None.

**Result** Returns the *maximum* character width (in pixels).

**Comments** This function returns the value of the `fRectHeight` field in the [FontType](#) structure for the current font. Because Palm OS does not support kerning, this value is the maximum width in pixels rather than the average width.

### FntBaseLine

**Purpose** Gets the distance from the top of the character cell to the baseline for the current font.

**Declared In** `Font.h`

**Prototype** `Int16 FntBaseLine (void)`

**Parameters** None.

**Result** Returns the ascent of the font (in pixels).

## Fonts

### *Font Functions*

---

## FntCharHeight

**Purpose** Gets the character height of the current font including accents and descenders.

**Declared In** Font.h

**Prototype** Int16 FntCharHeight (void)

**Parameters** None.

**Result** Returns the height of the characters in the current font, expressed in pixels.

## FntCharsInWidth

**Purpose** Finds the length in bytes of the characters from a specified string that fit within a passed width.

**Declared In** Font.h

**Prototype** void FntCharsInWidth (Char const \*string,  
Int16 \*stringWidthP, Int16 \*stringLengthP,  
Boolean \*fitWithinWidth)

**Parameters** -> string A pointer to the character string.

<- stringWidthP

The maximum width to allow (in pixels). Upon return, contains the actual width allowed. Note that this value does not include any trailing spaces or tabs, which are stripped by this function.

<-> stringLengthP

The maximum length of text to allow, in bytes (assumes current font). Upon return, contains the number of bytes of text that can appear within the width. Note that this value does not include any trailing space or tabs, which are stripped by this function.

<- fitWithinWidth

Upon return, `false` if the string is considered truncated, `true` if it isn't.

**Result** Returns nothing.

**Comments** Spaces and tabs at the end of a string are ignored and removed. If the string fits within the specified width after spaces and tabs are removed, the `fitWithinWidth` value contains `true`. Characters after a carriage return are ignored, and the string is considered truncated.

This function is specifically designed for the code used to draw text fields. Consider using [FntWidthToOffset](#) in your application code instead, particularly if you do not want the special processing of trailing spaces, tabs, and carriage returns.

## FntCharsWidth

**Purpose** Gets the width of the specified character string. The **missing character symbol** (an open rectangle) is substituted for any character that does not exist in the current font.

**Declared In** Font.h

**Prototype** Int16 FntCharsWidth (Char const \*chars,  
Int16 len)

**Parameters** -> chars      Pointer to a string of characters.

## Fonts

### Font Functions

---

-> len                    Length in bytes of the string.

**Result** Returns the width of the string, in pixels.

**Comments** Like all functions that work with strings, this function returns correct results for strings with multi-byte characters as well as strings with only single-byte characters.

**See Also** [FntCharWidth](#)

## FntCharWidth

**Purpose** Gets the width of the specified character. If the specified character does not exist within the current font, the missing character symbol is substituted.

**Declared In** Font.h

**Prototype** Int16 FntCharWidth (Char ch)

**Parameters** -> ch                    Character whose width is needed.

**Result** Returns the width of the specified character (in pixels).

**Comments** FntCharWidth works with single-byte characters only. To determine the pixel width of a single-byte character or a multi-byte character, use [FntWCharWidth](#) instead of this function in Palm OS 4.0 and higher. Alternatively, for compatibility with earlier versions of Palm OS, link with the PalmOSGlue library and call FntGlueWCharWidth. For more information, see [Chapter 75, "PalmOSGlue Library."](#)

**See Also** [FntCharsWidth](#)

## FntDefineFont

**Purpose** Makes a custom font available to your application.

**Declared In** `Font.h`

**Prototype** `Err FntDefineFont (FontID font, FontPtr fontP)`

**Parameters**

-> font	A value greater than or equal to <code>fntAppFontCustomBase</code> that identifies the custom font to the system. Values less than that are reserved for system use. Note that font IDs are 8-bit unsigned values and so must be less than 256. See <a href="#">Font ID</a> .
-> fontP	Pointer to the custom font resource to be used by this function. This resource must remain locked until the calling application undefines the custom font or quits.

**Result** `errNone` No error

`memErrNotEnoughSpace`  
Insufficient dynamic heap space

**Comments** The custom font is available only when the application that called this function is running; when the application quits, the custom font is uninstalled automatically.

The font this function specifies is not available at build time; as a result, some UI elements—labels, for example—cannot determine their bounds automatically as they do when using the built-in fonts.

Before you use this function, you must load the font resource from the database and obtain a pointer to it. See “[Creating Custom Fonts](#)” on page 275 of the *Palm OS Programmer’s Companion*, vol. I for more information.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

Palm OS 4.0 and later do not allow a NULL value for the `fontP` parameter; therefore, you cannot undefine a font using

## Fonts

### Font Functions

---

`FntDefineFont (myFontID, NULL)`. Earlier versions of Palm OS allowed the NULL value. Note that there is usually no need to undefine a font.

Palm OS 3.0 and 3.1 have problems with font ID values greater than 131. These problems are fixed in later releases of Palm OS.

**See Also** [FontSelect](#), [FntSetFont](#)

## FntDescenderHeight

**Purpose** Gets the height of a character's descender in the current font. The height of a descender is the distance between the baseline and the bottom of the character cell.

**Declared In** `Font.h`

**Prototype** `Int16 FntDescenderHeight (void)`

**Parameters** None.

**Result** Returns the height of a descender, expressed in pixels.

## FntGetFont

**Purpose** Gets the font ID of the current font.

**Declared In** `Font.h`

**Prototype** `FontID FntGetFont (void)`

**Parameters** None.

**Result** Returns the ID of the current font.

**Comments** The current font is the font stored in the draw state. It is used when drawing characters directly onto the screen. Most user interface

elements, such as fields, tables, labels, and buttons, do not use the current font.

**See Also** [FntSetFont](#), [FntGetFontPtr](#), [FontID](#)

## FntGetFontPtr

**Purpose** Gets a pointer to the current font.

**Declared In** Font.h

**Prototype** FontPtr FntGetFontPtr (void)

**Parameters** None.

**Result** Returns a pointer to the current font.

**Comments** The current font is the font stored in the draw state. It is used when drawing characters directly onto the screen. Most user interface elements, such as fields, tables, labels, and buttons, do not use the current font.

**See Also** [FntGetFont](#)

## FntGetScrollValues

**Purpose** Gets the values needed to update a scroll bar based on a specified string and the position within the string.

**Declared In** Font.h

**Prototype** void FntGetScrollValues (Char const \*chars,  
                  UInt16 width, UInt16 scrollPos, UInt16 \*linesP,  
                  UInt16 \*topLine)

**Parameters** -> chars                   A null-terminated string.

## Fonts

### Font Functions

---

-> width	The width of a line of text in the display, given in pixels.
-> scrollPos	The byte offset of the first character displayed on the topmost line.
<- linesP	Number of lines required to display the string.
<- topLine	The line of text that is the topmost visible line. Line numbering starts with 0.

**Result** Returns nothing. Stores the number of lines of text in `linesP` and the top visible line in `topLine`.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

**See Also** [FldGetScrollValues](#)

## FntIsAppDefined

**Purpose** Macro that returns `true` if the font is defined by the application or `false` if it is defined by the system.

**Declared In** Font.h

**Prototype** FntIsAppDefined (fnt)

**Parameters** -> fnt                    The [Font ID](#) of a font.

**Result** Boolean that indicates if the font is an application-defined font. Returns `true` if application-defined, `false` if system-defined.

## FntLineHeight

**Purpose** Gets the height of a line in the current font. The height of a line is the height of the character cell plus the space between lines (the external leading).

**Declared In** Font.h

**Prototype** Int16 FntLineHeight (void)

**Parameters** None.

**Result** Returns the height in pixels of a line in the current font.

## FntLineWidth

**Purpose** Gets the width of the specified line of text, taking tab characters into account. The function assumes that the characters passed are left-aligned and that the first character in the string is the first character drawn on a line. In other words, this routine doesn't work for characters that don't start at the beginning of a line.

**Declared In** Font.h

**Prototype** Int16 FntLineWidth (Char const \*pChars,  
                  UInt16 length)

**Parameters** -> pChars                 Pointer to a string of characters.  
                  -> length                 Length in bytes of the string.

**Result** Returns the line width (in pixels).

## Fonts

### *Font Functions*

---

## FntSetFont

**Purpose** Sets the current font.

**Declared In** Font.h

**Prototype** FontID FntSetFont (FontID font)

**Parameters** -> font ID of the font to make the current font.

**Result** Returns the ID of the previous font.

**Comments** If the specified font ID is invalid, this function sets the current font to stdFont.

The current font is the font stored in the draw state. It is used when drawing characters directly onto the screen. Most user interface elements, such as fields, tables, labels, and buttons, do not use the current font. To set the font for one of these elements, check the API for that element. If the element's API doesn't have a function to set the font programmatically, check the PalmOSGlue library.

**See Also** [FntGetFont](#)

## FntWCharWidth

**Purpose** Gets the width of the specified character. If the specified character does not exist within the current font, the missing character symbol is substituted.

**Declared In** Font.h

**Prototype** Int16 FntWCharWidth (WChar iChar)

**Parameters** -> iChar Character whose width is needed.

**Result** Returns the width of the specified character (in pixels).

<b>Comments</b>	FntWCharWidth works with both single-byte characters and multi-byte characters. However, you should always pass a WChar variable to this function rather than a Char to avoid sign extension problems on values 0x80 and higher.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present. If you want to use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call FntGlueWCharWidth. For more information, see <a href="#">Chapter 75, "PalmOSGlue Library."</a>
<b>See Also</b>	<a href="#">FntCharWidth</a>

## FntWidthToOffset

<b>Purpose</b>	Given a pixel position, gets the offset of the character displayed at that location.								
<b>Declared In</b>	Font.h								
<b>Prototype</b>	Int16 FntWidthToOffset (Char const *pChars, UIInt16 length, Int16 pixelWidth, Boolean *leadingEdge, Int16 *truncWidth)								
<b>Parameters</b>	<table><tr><td>-&gt; pChars</td><td>Pointer to the character string.</td></tr><tr><td>-&gt; length</td><td>The length in bytes of pChars.</td></tr><tr><td>-&gt; pixelWidth</td><td>A horizontal pixel offset from the beginning of the string.</td></tr><tr><td>&lt;- leadingEdge</td><td>Set to true if the pixel position pixelWidth falls on the left side of the character. Pass NULL for this parameter if you don't need this information.</td></tr></table>	-> pChars	Pointer to the character string.	-> length	The length in bytes of pChars.	-> pixelWidth	A horizontal pixel offset from the beginning of the string.	<- leadingEdge	Set to true if the pixel position pixelWidth falls on the left side of the character. Pass NULL for this parameter if you don't need this information.
-> pChars	Pointer to the character string.								
-> length	The length in bytes of pChars.								
-> pixelWidth	A horizontal pixel offset from the beginning of the string.								
<- leadingEdge	Set to true if the pixel position pixelWidth falls on the left side of the character. Pass NULL for this parameter if you don't need this information.								

## Fonts

### Font Functions

---

<- truncWidth      The width of the text (in pixels) up to but not including the returned offset. Pass NULL for this parameter if you don't need this information.

**Result**      Returns the byte offset into pChars of the character that contains the pixel offset pixelWidth. If pixelWidth is past the right edge of the string, the function returns the byte offset past the last character in pChars, and truncWidth contains the width required to display the entire string.

**Compatibility**      Implemented only if [3.1 New Feature Set](#) is present. If you want to use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call FntGlueWidthToOffset. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

## FntWordWrap

**Purpose**      Given a string, determines how many bytes of text can be displayed within the specified width with a line break at a tab or space character.

**Declared In**      Font.h

**Prototype**      `UInt16 FntWordWrap (Char const *chars,  
                  UInt16 maxWidth)`

**Parameters**      -> chars      A pointer to a null-terminated string.  
                  -> maxWidth      The maximum line width in pixels.

**Result**      Returns the length of the line, in bytes. If the entire string cannot be displayed within maxWidth, the value that this function returns specifies the offset where the line should be broken, which is typically following a space, tab, or line-feed character.

**Compatibility**      Implemented only if [2.0 New Feature Set](#) is present.

**See Also**      [FltWordWrap](#)

## FntWordWrapReverseNLines

**Purpose** Word wraps a text string backwards by the number of lines specified. The character position of the start of the first line and the number of lines that are actually word wrapped are returned.

**Declared In** Font.h

**Prototype** void FntWordWrapReverseNLines  
(Char const \*const chars, UInt16 maxWidth,  
UInt16 \*linesToScrollP, UInt16 \*scrollPosP)

**Parameters**

-> chars	A pointer to a null-terminated string.
-> maxWidth	The maximum line width in pixels.
<-> linesToScrollP	The number of lines to scroll. Upon return, contains the number of lines that were scrolled.
<-> scrollPosP	The byte offset of the first character displayed on the topmost line. Upon return, contains the first character after wrapping.

**Result** Returns nothing. Stores the first character after wrapping and the number of lines scrolled in scrollPosP and linesToScrollP.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## Fonts

### Font Functions

---

## FontSelect

**Purpose** Displays a dialog from which the user can choose one of the system-supplied fonts, and returns a `FontID` value representing the user's choice.

**Declared In** `FontSelect.h`

**Prototype** `FontID FontSelect (FontID fontID)`

**Parameters** `-> fontID` A `FontID` value specifying the font to be highlighted as the default choice in the dialog box that this function displays. This value must be one of the following system-supplied constants:

`stdFont`  
Standard plain text font.

`boldFont`  
Bold version of `stdFont`.

`largeFont`  
A large plain text font (Japanese devices only).

`largeBoldFont`  
Larger version of `boldFont`.

**Result** Returns a `FontID` value representing the font that the user chose.

**Comments** When your application launches for the first time, it should determine the system's default font. The default font varies based on locale. You can use [`FntGlueGetDefaultFontID`](#) from the PalmOSGlue library to determine the default font as follows:

```
fntID =  
FntGlueGetDefaultFontID(defaultSystemFont);
```

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [FntGetFont](#), [FntSetFont](#)

# Graffiti Manager

---

This chapter provides reference material for the Graffiti® manager. The Graffiti manager API is declared in the header file `Graffiti.h`.

For more information on the Graffiti manager, see “[The Graffiti Manager](#)” on page 60 of the *Palm OS Programmer’s Companion*, vol. I.

## Graffiti Manager Functions

### GrfAddMacro

**Purpose** Add a macro to the macro list.

**Declared In** `Graffiti.h`

**Prototype** `Err GrfAddMacro (const Char *nameP,  
                  UInt8 *macroDataP, UInt16 dataLen)`

**Parameters** `nameP` Name of macro.  
`macroDataP` Data of macro.  
`dataLen` Size of macro data in bytes.

**Result** Returns 0 if no error; returns `grfErrNoMacros`, `grfErrMacroPtrTooSmall`, `dmErrNotValidRecord`, `dmErrWriteOutOfBounds` if an error occurs.

**See Also** [GrfGetMacro](#), [GrfGetMacroName](#), [GrfDeleteMacro](#)

## Graffiti Manager

### *Graffiti Manager Functions*

---

## GrfAddPoint

**Purpose** Add a point to the Graffiti point buffer.

**Declared In** Graffiti.h

**Prototype** Err GrfAddPoint (PointType \*pt)

**Parameters** pt Pointer to point buffer.

**Result** Returns 0 if no error; returns grfErrPointBufferFull if an error occurs.

**See Also** [GrfFlushPoints](#)

## GrfCleanState

**Purpose** Remove any temporary shifts from the dictionary state.

**Declared In** Graffiti.h

**Prototype** Err GrfCleanState (void)

**Parameters** None

**Result** Returns 0 if no error, or grfErrNoDictionary if an error occurs.

**See Also** [GrfInitState](#)

## GrfDeleteMacro

**Purpose** Delete a macro from the macro list.

**Declared In** Graffiti.h

**Prototype** Err GrfDeleteMacro (UInt16 index)

**Parameters** index Index of the macro to delete.

**Result** Returns 0 if no error, or grfErrNoMacros, grfErrMacroNotFound if an error occurs.

**See Also** [GrfAddMacro](#)

## GrfFilterPoints

**Purpose** Filter the points in the Graffiti point buffer.

**Declared In** Graffiti.h

**Prototype** Err GrfFilterPoints (void)

**Parameters** None.

**Result** Always returns 0.

**See Also** [GrfMatch](#)

## Graffiti Manager

### *Graffiti Manager Functions*

---

## GrfFindBranch

**Purpose** Locate a branch in the Graffiti dictionary by flags.

**Declared In** Graffiti.h

**Prototype** Err GrfFindBranch (UInt16 flags)

**Parameters** flags Flags of the branch you're searching for.

**Result** Returns 0 if no error, or grfErrNoDictionary or grfErrBranchNotFound if an error occurs.

**See Also** [GrfCleanState](#), [GrfInitState](#)

## GrfFlushPoints

**Purpose** Dispose of all points in the Graffiti point buffer.

**Declared In** Graffiti.h

**Prototype** Err GrfFlushPoints (void)

**Parameters** None.

**Result** Always returns 0.

**See Also** [GrfAddPoint](#)

## GrfGetAndExpandMacro

**Purpose** Look up and expand a macro in the current macros.

**Declared In** Graffiti.h

**Prototype** Err GrfGetAndExpandMacro (Char \*nameP,  
UInt8 \*macroDataP, UInt16 \*dataLenP)

**Parameters**

nameP	Name of macro to look up.
macroDataP	Macro contents returned here.
dataLenP	On entry, size of macroDataP buffer; on exit, number of bytes in macro data.

**Result** Returns 0 if no error, or grfErrNoMacros or grfErrMacroNotFound if an error occurs.

**See Also** [GrfAddMacro](#), [GrfGetMacro](#)

## GrfGetGlyphMapping

**Purpose** Look up a glyph in the dictionary and return the text.

**Declared In** Graffiti.h

**Prototype** Err GrfGetGlyphMapping (UInt16 glyphID,  
UInt16 \*flagsP, void \*dataPtrP, UInt16 \*dataLenP,  
UInt16 \*uncertainLenP)

**Parameters**

glyphID	Glyph ID to look up.
flagsP	Returned dictionary flags.
dataPtrP	Where returned text goes.
dataLenP	On entry, size of dataPtrP; on exit, number of bytes returned.

## Graffiti Manager

### *Graffiti Manager Functions*

---

uncertainLenP    Return number of uncertain characters in text.

**Result**    Returns 0 if no error, or grfErrNoDictionary or grfErrNoMapping if an error occurs.

**See Also**    [GrfMatch](#)

## GrfGetMacro

**Purpose**    Look up a macro in the current macros.

**Declared In**    Graffiti.h

**Prototype**    Err GrfGetMacro (Char \*nameP, UInt8 \*macroDataP,  
                  UInt16 \*dataLenP)

**Parameters**    nameP                 Name of macro to lookup.  
                  macroDataP          Macro contents returned here.  
                  dataLenP             On entry: size of macroDataP buffer. On exit:  
                                       number of bytes in macro data.

**Result**    Returns 0 if no error or grfErrNoMacros,  
                  grfErrMacroNotFound.

**See Also**    [GrfAddMacro](#)

## GrfGetMacroName

**Purpose**    Look up a macro name by index.

**Declared In**    Graffiti.h

**Prototype**    Err GrfGetMacroName (UInt16 index, Char \*nameP)

**Parameters**    index                 Index of macro.

nameP                    Name returned here.

**Result**    Returns 0 if no error, or grfErrNoMacros or grfErrMacroNotFound if an error occurs.

**See Also**    [GrfAddMacro](#), [GrfGetMacro](#)

## GrfGetNumPoints

**Purpose**    Return the number of points in the point buffer.

**Declared In**    Graffiti.h

**Prototype**    Err GrfGetNumPoints (UInt16 \*numPtsP)

**Parameters**    numPtsP                    Returned number of points.

**Result**    Always returns 0.

**See Also**    [GrfAddPoint](#)

## GrfGetPoint

**Purpose**    Return a point out of the Graffiti point buffer.

**Declared In**    Graffiti.h

**Prototype**    Err GrfGetPoint (UInt16 index, PointType \*pointP)

**Parameters**    index                    Index of the point to get.  
                    pointP                Returned point.

**Result**    Returns 0 if no error, or grfErrBadParam if an error occurs.

**See Also**    [GrfAddPoint](#), [GrfGetNumPoints](#)

## GrfGetState

**Purpose** Return the current Graffiti shift state.

**Declared In** Graffiti.h

**Prototype** Err GrfGetState (Boolean \*capsLockP,  
Boolean \*numLockP, UInt16 \*tempShiftP,  
Boolean \*autoShiftedP)

**Parameters**

capsLockP	Returns true if caps lock on.
numLockP	Returns true if num lock on.
tempShiftP	Current temporary shift.
autoShiftedP	Returns TRUE if shift not set by the user but by the system, for example, at the beginning of a line.

**Result** Always returns 0.

**Compatibility** Palm OS® 2.0 and later has more user-friendly auto shifting. It uses an upper case letter under these conditions:

- after an empty field
- after a period or other sentence terminator (such as ? or !).
- after two spaces

**See Also** [GrfSetState](#)

## GrfInitState

**Purpose** Reinitialize the Graffiti dictionary state.

**Declared In** Graffiti.h

**Prototype** Err GrfInitState (void)

**Parameters** None.

**Result** Always returns 0.

**See Also** [GrfGetState](#), [GrfSetState](#)

## GrfMatch

**Purpose** Recognize the current stroke in the Graffiti point buffer and return with the recognized text.

**Declared In** Graffiti.h

**Prototype** Err GrfMatch (UInt16 \*flagsP, void \*dataPtrP,  
                  UInt16 \*dataLenP, UInt16 \*uncertainLenP,  
                  GrfMatchInfoPtr matchInfoP)

<b>Parameters</b>	flagsP	Glyph flags are returned here.
	dataPtrP	Return text is placed here.
	dataLenP	Size of dataPtrP on exit; number of characters returned on exit.
	uncertainLenP	Return number of uncertain characters.
	matchInfoP	Array of grfMaxMatches, or NULL.

**Result** Returns 0 if no error, or grfErrNoGlyphTable,  
grfErrNoDictionary, or grfErrNoMapping if an error occurs.

**See Also** [GrfAddPoint](#), [GrfFlushPoints](#)

## Graffiti Manager

### *Graffiti Manager Functions*

---

## GrfMatchGlyph

**Purpose** Recognize the current stroke as a glyph.

**Declared In** Graffiti.h

**Prototype** Err GrfMatchGlyph (GrfMatchInfoPtr matchInfoP,  
Int16 maxUncertainty, UInt16 maxMatches)

**Parameters** matchInfoP Pointer to array of matches to fill in.

maxUncertainty Maximum number of errors to tolerate.

maxMatches Size of matchInfoP array.

**Result** Returns 0 if no error, or grfErrNoGlyphTable if an error occurs.

**See Also** [GrfMatch](#)

## GrfProcessStroke

**Purpose** Translate a stroke to keyboard events using Graffiti.

**Declared In** Graffiti.h

**Prototype** Err GrfProcessStroke (PointType \*startPtP,  
PointType \*endPtP, Boolean upShift)

<b>Parameters</b>	startPtP	Start point of stroke.
	endPtP	End point of stroke.
	upShift	Set to true to feed an artificial upshift into the engine.

**Result** Returns 0 if recognized.

**Comments** Called by SysHandleEvent when a penUpEvent is detected in the writing area. This routine recognizes the stroke and sends the recognized characters into the key queue. It also flushes the stroke out of the pen queue after recognition.

**See Also** [SysHandleEvent](#)

## GrfSetState

**Purpose** Set the current shift state of Graffiti.

**Declared In** Graffiti.h

**Prototype** Err GrfSetState (Boolean capsLock,  
Boolean numLock, Boolean upperShift)

<b>Parameters</b>	capsLock	Set to true to turn on caps lock.
	numLock	Set to true to turn on num lock.
	upperShift	Set to true to put into upper shift.

**Result** Always returns 0.

**See Also** [GrfGetState](#)

## **Graffiti Manager**

### *Graffiti Manager Functions*

---

# Helper API

---

This chapter describes the Helper API declared in the header files `Helper.h` and `HelperServiceClass.h`. The Helper API is used when an application broadcasts a [`sysNotifyHelperEvent`](#) to all interested parties. The broadcaster of the notification and the notification clients (called **helpers**) use the Helper APIs to communicate with each other. The chapter discusses the following topics:

- [Helper Data Structures](#)
- [Helper Constants](#)

For more information on using the Helper API, see the section “[Helper Notifications](#)” on page 38 of the *Palm OS Programmer’s Companion*, vol. I.

## Helper Data Structures

### **HelperNotifyEnumerateListType**

The `HelperNotifyEnumerateListType` provides the broadcaster of the helper notification with information about the services that the helper can provide. This structure is used as the data field of the [`HelperNotifyEventType`](#) structure when the action code is `kHelperNotifyActionCodeEnumerate`.

```
typedef struct HelperNotifyEnumerateListTypeTag
{
    struct HelperNotifyEnumerateListTypeTag
        *nextP;
    Char helperAppName [kHelperAppMaxNameSize] ;
    Char actionName [kHelperAppMaxActionNameSize] ;
    UInt32 helperAppID;
    UInt32 serviceClassID;
} HelperNotifyEnumerateListType;
```

## Helper API

### Helper Data Structures

---

Note that the helper allocates this structure and then adds it to the linked list of structures pointed to by `notifyDetailsP->data.enumerateP` in the [SysNotifyParamType](#) that is sent to the helper. The helper should allocate one structure per supported service.

Even though the helper allocates this structure, the helper is not responsible for freeing the structure. Instead, the application that broadcast the notification must free the structure.

#### Field Descriptions

<code>nextP</code>	A pointer to the next element in the list or NULL to signal the end of the list.
<code>helperAppName</code>	A null-terminated string containing the name of the helper application, suitable for display in the user interface. If more than one application can perform the same service, this name is displayed as one of the choices in a pop-up list.
<code>actionName</code>	A null-terminated string containing the name of the service that can be performed, suitable for display in the user interface. The action name should be short enough to display on a button.
<code>helperAppID</code>	The helper's creator ID or any other ID that uniquely identifies the helper.
<code>serviceClassID</code>	The ID of the service that the helper performs. See <a href="#"><u>Helper Service Class IDs</u></a> .

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## HelperNotifyEventType

The HelperNotifyEventType structure contains all data associated with a helper notification ([sysNotifyHelperEvent](#)). A pointer to this structure is passed as the notifyDetailsP field in the [SysNotifyParamType](#) for that notification.

```
typedef struct HelperNotifyEventTypeTag {  
    UInt16 version;  
    HelperNotifyActionCodeType actionCode;  
    union {  
        struct HelperNotifyEnumerateListTypeTag  
            *enumerateP;  
        struct HelperNotifyValidateTypeTag  
            *validateP;  
        struct HelperNotifyExecuteTypeTag  
            *executeP;  
    } data;  
} HelperNotifyEventType;
```

### Field Descriptions

**version**      The version number for this structure. The current version is 1.

**actionCode**    The action that the helper application should perform. See [Table 35.1](#).

**data**          Data specific to the action code. See [Table 35.1](#).

The HelperNotifyEventType structure specifies which action is to be performed and contains data necessary for that action. All actions have some common data. Actions also have data specific to that action. The specific data uses a union that is part of the HelperNotifyEventType structure.

## Helper API

### Helper Data Structures

---

**Table 35.1 Helper action codes**

Action Code	data Type	Description
kHelperNotify ActionCode Enumerate	<a href="#">HelperNotifyEnumerateListType</a>	Send a list of available services.
kHelperNotify ActionCode Validate	<a href="#">HelperNotifyValidateType</a>	Validate the input data for the specified service.
kHelperNotify ActionCode Execute	<a href="#">HelperNotifyExecuteType</a>	Perform the specified service.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

### HelperNotifyExecuteType

The HelperNotifyExecuteType structure identifies the service to perform and contains the data necessary to perform that service. This structure is used as the data field of the [HelperNotifyEventType](#) structure when the action code is kHelperNotifyActionCodeExecute.

```
typedef struct HelperNotifyExecuteTypeTag {  
    UInt32    serviceClassID;  
    UInt32    helperAppID;  
    Char *dataP;  
    Char *displayedName;  
    void *detailsP;  
    Err err;  
} HelperNotifyExecuteType;
```

## Field Descriptions

serviceClassID	The ID of the service to be performed. See <a href="#">Helper Service Class IDs</a> .
helperAppID	The unique ID of the helper; a value of 0 indicates that any available helper for the specified service class should perform the service.
dataP	A null-terminated string specific to this service, such as a phone number for the dial service or an email address for the email service. See <a href="#">Table 35.2</a> . Multiple fields must be separated by semicolons (;).
displayName	A null-terminated string containing an optional, human-readable description of the string in dataP. For example, if dataP contains a phone number, this field might contain the name of the person at that number.
detailsP	A pointer to a data structure containing extra information that this service requires. See <a href="#">Table 35.2</a> . If the service does not require extra information, this field is NULL.
err	An error code that indicates if the service was performed successfully or not. If the service was successful, this field contains errNone, and the handled field in the notification data structure should be set to true.

The following table lists the Palm OS-defined values for the service class ID and for each service, shows what value dataP contains and what type of structure detailsP points to.

## Helper API

### Helper Data Structures

---

**Table 35.2 HelperNotifyExecuteType values**

Service Class ID	dataP Value	detailsP Value
kHelper ServiceClassID VoiceDial	The telephone number to dial	NULL
kHelper ServiceClassID EMail	The email address to which the message is to be sent	<a href="#">HelperServiceEMailDetailsType</a>
kHelper ServiceClassID SMS	The SMS mailbox number to which the message is to be sent	<a href="#">HelperServiceSMSDetailsType</a>
kHelper ServiceClassID Fax	The fax number to which the fax is to be sent	NULL

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## HelperNotifyValidateType

The HelperNotifyValidateType structure identifies a service that should be validated and the helper that should validate it. This structure is used as the data field of the [HelperNotifyEventType](#) structure when the action code is kHelperNotifyActionCodeValidate.

```
typedef struct HelperNotifyValidateTypeTag {  
    UInt32 serviceClassID;  
    UInt32 helperAppID;  
} HelperNotifyValidateType;
```

### Field Descriptions

`serviceClassID` The ID of the service to be validated. See [Helper Service Class IDs](#).

`helperAppID` The creator ID of the helper application. 0 indicates that any available helper for the specified service should respond. If nonzero, only the helper with the matching creator ID should respond.

The helper returns `true` in the `handled` field of the [SysNotifyParamType](#) structure to indicate that the service can be performed or `false` to indicate that the service cannot be performed.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## HelperServiceEMailDetailsType

The `HelperServiceEMailDetailsType` structure provides additional data for the email service. It is used as the `detailsP` field in the [HelperNotifyExecuteType](#) when the service class ID is `kHelperServiceClassIDEMail`.

```
typedef struct _HelperServiceEMailDetailsType
{
    UInt16 version;
    Char *cc;
    Char *subject;
    Char *message;
} HelperServiceEMailDetailsType;
```

### Field Descriptions

`version` The version number for this structure. The current version is 1.

`cc` A null-terminated string containing an email address that should be sent a carbon copy of the message. Multiple addresses are separated by a semi-colon (;). May be `NULL` if there are no email addresses to carbon copy.

subject	A null-terminated string containing the subject line. May be NULL.
message	Initial message body string or NULL.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## HelperServiceSMSDetailsType

The HelperServiceSMSDetailsType structure provides the SMS message to be sent. It is used as the detailsP field in the [HelperNotifyExecuteType](#) when the service class ID is kHelperServiceClassIDSMS.

```
typedef struct _HelperServiceSMSDetailsType {  
    UInt16 version;  
    Char *message;  
} HelperServiceSMSDetailsType;
```

### Field Descriptions

version The version number for this structure. The current version is 1.

message A null-terminated string containing the body of the message to be sent, or NULL.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## Helper Constants

### Helper Service Class IDs

The header file HelperServiceClass.h defines the constants listed in [Table 35.3](#) as service class IDs. Third party developers may define their own service classes. To do so, you must register a 32-bit identifier with PalmSource, Inc. on this web site:

<http://www.palmos.com/dev/creatorid/>

Alternatively, you can use a creator ID that you already own.

**Table 35.3 Service class ID constants**

<b>Constant</b>	<b>Value</b>	<b>Description</b>
kHelperServiceClassIDVoiceDial	'voic'	Dial a phone number for a voice telephone call.
kHelperServiceClassIDEMail	'mail'	Send an email message.
kHelperServiceClassIDSMS	'sms_'	Send an SMS message.
kHelperServiceClassIDFax	'fax_'	Send a fax.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## **Helper API**

### *Helper Constants*

---

# Key Manager

---

This chapter provides reference material for the key manager. The key manager API is declared in the header file KeyMgr.h.

For more information on the key manager, see “[The Key Manager](#)” on page 62 of the *Palm OS Programmer’s Companion*, vol. I.

## Key Manager Functions

### **KeycurrentState**

**Purpose**    Return bit field with bits set for each key that is currently depressed.

**Declared In**    KeyMgr.h

**Prototype**    `UInt32 KeycurrentState (void)`

**Parameters**    None.

**Result**    Returns a `UInt32` with bits set for keys that are depressed. See `keyBitPower`, `keyBitPageUp`, `keyBitPageDown`, etc., in KeyMgr.h.

**Comments**    Called by applications that need to poll the keys.

**See Also**    [KeyRates](#)

## Key Manager

### *Key Manager Functions*

---

## KeyRates

**Purpose** Get or set the key repeat rates.

**Declared In** KeyMgr.h

**Prototype** Err KeyRates (Boolean set, UInt16\* initDelayP,  
UInt16\* periodP, UInt16\* doubleTapDelayP,  
Boolean\* queueAheadP)

**Parameters**

set	If <code>true</code> , settings are changed; if <code>false</code> , current settings are returned.
initDelayP	Initial delay in ticks for a auto-repeat event.
periodP	Auto-repeat rate specified as period in ticks.
doubleTapDelayP	Maximum double-tap delay, in ticks.
queueAheadP	If <code>true</code> , auto-repeating keeps queueing up key events if the queue has keys in it. If <code>false</code> , auto-repeat doesn't enqueue keys unless the queue is already empty.

**Result** Returns 0 if no error.

**See Also** [KeyCurrentState](#)

## KeySetMask

**Purpose** Specify which keys generate keyDownEvents.

You can specify this either by using this function or by using the poweredOnKeyMask modifier.

**Declared In** KeyMgr.h

**Prototype** UInt32 KeySetMask (UInt32 keyMask)

**Parameters** keyMask Mask with bits set for those keys to generate keyDownEvents for.

**Result** Returns the old key Mask.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## **Key Manager**

### *Key Manager Functions*

---

# Locale Manager

---

This chapter describes the Locale Manager API as described in the header files `LocaleMgr.h`, `Localize.h`, and `PalmLocale.h`. It discusses the following topics:

- [Locale Manager Data Types](#)
- [Locale Manager Constants](#)
- [Locale Manager Functions](#)

For more information on the Locale Manager, see the chapter “[Localized Applications](#)” on page 363 of the *Palm OS Programmer’s Companion*, vol. I.

## Locale Manager Data Types

### CountryType

The `CountryType` defines a country code. The [Country Constants](#) in `PalmLocale.h` define the possible values for `CountryType` variables.

```
typedef UInt8 CountryType;
```

#### Compatibility

Prior to version 4.0, `CountryType` was an enum in `Preferences.h` that defined only 33 country codes. The Palm OS® 4.0 definition of `CountryType` is compatible with the previous definition.

### LanguageType

The `LanguageType` defines a language code. The [Language Constants](#) in `PalmLocale.h` define the possible values for `LanguageType` variables.

## Locale Manager

### Locale Manager Data Types

---

```
typedef UInt8 LanguageType;
```

#### Compatibility

The LanguageType definition was added in Palm OS 3.5. Prior to version 4.0, LanguageType was an enum in Preferences.h that defined only eight language codes. The Palm OS 4.0 definition of LanguageType is compatible with the previous definition.

## LmLocaleType

The LmLocaleType struct defines the country and language used in a locale.

```
struct _LmLocaleType {  
    UInt16        language;  
    UInt16        country;  
};  
typedef struct _LmLocaleType LmLocaleType;
```

#### Field Descriptions

**language** One of the [Language Constants](#). This value identifies the language spoken in the current locale.

**country** One of the [Country Constants](#). This value identifies the locale's country, which helps to identify the language dialect. For example, a language of cEnglish specifies a different dialect if the country is cUnitedKingdom than if it is cUnitedStates.

Note that the language and country fields are type UInt16 instead of LanguageType and CountryType.

#### Compatibility

The LmLocaleType is defined only if [4.0 New Feature Set](#) is present. It supersedes the OmLocaleType that was introduced with Palm OS 3.5. LmLocaleType is bit-compatible with OmLocaleType.

## NumberFormatType

The NumberFormatType enum specifies how numbers are formatted. You can pass a NumberFormatType value to

[LocGetNumberSeparators](#) and receive the appropriate separator characters for thousands and decimals.

```
typedef enum {
    nfCommaPeriod,
    nfPeriodComma,
    nfSpaceComma,
    nfApostrophePeriod,
    nfApostropheComma
} NumberFormatType;
```

### Value Descriptions

nfCommaPeriod	Uses a comma (,) as the thousands separator and a period (.) as the decimal separator.
nfPeriodComma	Uses a period as the thousands separator and a comma as the decimal separator.
nfSpaceComma	Uses a space ( ) as the thousands separator and a comma as the decimal separator.
nfApostrophePeriod	Uses an apostrophe (') as the thousands separator and a period as the decimal separator.
nfApostropheComma	Uses an apostrophe as the thousands separator and a comma as the decimal separator.

## Locale Manager Constants

### Character Encoding Constants

The PalmLocale.h file defines several character encoding constants that are used as values of [CharEncodingType](#) variables. The character encoding constants generally follow the format:

charEncodingName

where *Name* is the name of the character encoding.

## Locale Manager

### *Locale Manager Constants*

---

The following table shows examples of the character encoding constants. For a complete list, see the `PalmLocale.h` file.

Constant	Description
<code>charEncodingUnknown</code>	Unknown to this version of Palm OS®
<code>charEncodingAscii</code>	ISO 646-1991
<code>charEncodingISO8859_1</code>	ISO 8859 Part 1 (also known as ISO Latin 1). This encoding is commonly used for the Roman alphabet
<code>charEncodingPalmLatin</code>	Palm OS version of Microsoft Windows code page 1252. This encoding is identical to code page 1252 with Palm-specific characters added in the control range.
<code>charEncodingShiftJIS</code>	Encoding for 0208-1990 with single-byte Japanese Katakana. This encoding is commonly used for Japanese alphabets.
<code>charEncodingPalmSJIS</code>	Palm OS version of Microsoft Windows code page 932. This encoding is identical to code page 932, with Palm-specific characters added in the control range and with a Yen symbol instead of the Reverse Solidus at location 0x5c.
<code>charEncodingCP1252</code>	Microsoft Windows extensions to ISO 8859 Part 1
<code>charEncodingCP932</code>	Microsoft Windows extensions to Shift JIS
<code>charEncodingUTF8</code>	Eight-bit safe encoding for Unicode

---

## Country Constants

The `PalmLocale.h` file defines several country constants that are used as values of `CountryType` variables. The country type constants have the following format:

`cCountryName`

where *CountryName* is the name of the country. There is one constant for each country identified in the ISO 3166 standard, which currently defines 239 countries.

The following table shows examples of the country type constants. For a complete list, see the `PalmLocale.h` file.

Constant	Description
<code>cAustralia</code>	Australia
<code>cAustria</code>	Austria
<code>cBelgium</code>	Belgium

## Language Constants

The `PalmLocale.h` file defines several language constants that are used as values of `LanguageType` variables. The language type constants have the following format:

`lLanguageName`

where *LanguageName* is the name of the language. There is one constant for each language specified in the ISO 639 standard, which currently defines 137 languages.

The following table shows examples of the language type constants. For a complete list, see the `PalmLocale.h` file.

Constant	Description
<code>lEnglish</code>	English
<code>lFrench</code>	French
<code>lGerman</code>	German

## Locale Manager

### Locale Manager Functions

---

## Locale Manager Size Constants

You can use the Locale Manager size constants to determine the size of strings to allocate for some of the locale settings.

---

**NOTE:** The variables in the table below do not count the terminating null character. Therefore, you need to allocate a string of size `kMaxCountryNameLen+1` to hold a country name, for example.

---

Constant	Value	Description
<code>kMaxCountryNameLen</code>	19	The maximum length of a country name string.
<code>kMaxCurrencyNameLen</code>	19	The maximum length of a currency name string.
<code>kMaxCurrencySymbolLen</code>	5	The maximum length of a currency symbol string.

## Locale Manager Functions

### LmGetLocaleSetting

**Purpose** Return the requested setting for a given locale.

**Declared In** `LocaleMgr.h`

**Prototype** `Err LmGetLocaleSetting (UInt16 iLocaleIndex,  
LmLocaleSettingChoice iChoice, void *oValue,  
UInt16 iValueSize)`

**Parameters**

-> <code>iLocaleIndex</code>	Index of the locale whose settings you want to retrieve.
-> <code>iChoice</code>	The setting you want to retrieve. This is a constant in the form <code>lmcChoiceSettingName</code> . See <a href="#">Table 37.1</a> for a list of possible values.

<- oValue      The value of the `iChoice` setting. The size of this buffer depends on the value of `iChoice`, as shown in [Table 37.1](#).

-> iValueSize      The size of the `oValue` buffer.

**Result**      Returns one of the following values:

errNone      Success.

lmErrBadLocaleIndex  
                  `iLocaleIndex` is out of range.

lmErrSettingDataOverflow  
                  The `oValue` buffer is too small to hold the setting's value.

lmErrBadLocaleSettingChoice  
                  The `iChoice` parameter contains an unknown or unsupported value.

**Comments**

This function accesses the private locale system resource and returns the requested information in the `oValue` parameter. The size and type of the `oValue` parameter depend on which setting you want to retrieve. [Table 37.1](#) lists and describes the possible settings and the type of data returned in `oValue` for each setting. For fixed-size values, make sure that `oValue` is no larger than the returned value.

This function returns the default settings for the locale. Users can override many of the locale settings using the Preferences application. Applications should always honor the user's preferences rather than the locale defaults. For this reason, it's recommended that if a corresponding system preference is available, you should check it instead (using [PrefGetPreference](#)). Use `LmGetLocaleSetting` only if you want to retrieve values that the user cannot override (such as the country name or currency symbol) or if you want to retrieve information about a locale other than the current locale.

## Locale Manager

### Locale Manager Functions

---

**Table 37.1 LmGetLocaleSetting choices and sizes**

ImChoice...	oValue Data Type	Description
CountryName	String buffer of size kMaxCountryNameLen+1 bytes	The name of the locale's country.
CurrencyName	String buffer of size kMaxCurrencyNameLen+1 bytes	The name of the currency used in this locale.
CurrencySymbol	String buffer of size kMaxCurrencySymbolLen+1 bytes	The symbol used to denote monetary values in this locale.
CurrencyDecimal Places	UInt16	The number of decimal places that monetary values are typically given.
DateFormat	<a href="#">DateFormatType</a>	The short date format used in this locale. For example:  95/12/31
Locale	<a href="#">LmLocaleType</a>	A structure containing the locale's language and country codes.
LongDateFormat	<a href="#">DateFormatType</a>	The long date format used in this locale. For example:  31 Dec 1995
MeasurementSystem	<a href="#">MeasurementSystemType</a>	The measurement system (metric system or English system) used in this locale.
NumberFormat	<a href="#">NumberFormatType</a>	The format used for numbers, with regards to the thousands separator and the decimal point, in this locale.

**Table 37.1 LmGetLocaleSetting choices and sizes (*continued*)**

<b>ImChoice...</b>	<b>oValue Data Type</b>	<b>Description</b>
TimeFormat	<a href="#">TimeFormatType</a>	The format used for time values in this locale.
TimeZone	Int16	The locale's default time zone given as the number of minutes east of Greenwich Mean Time (GMT).
UniqueCurrencySymbol	String buffer of size kMaxCurrencySymbolLen+1 bytes	A unique symbol for monetary values.  For example, the symbol \$ is used both for US dollars and Portuguese escudos. The unique currency symbol for US dollars is US\$.
WeekStartDay	UInt16	The first day of the week (Sunday or Monday) in this locale. Days of the week are numbered from 0 to 6 starting with Sunday = 0.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call [LmGlueGetLocaleSetting](#). For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [LmGetNumLocales](#), [LmLocaleToIndex](#)

## Locale Manager

### *Locale Manager Functions*

---

## LmGetNumLocales

**Purpose** Return the number of known locales.

**Declared In** LocaleMgr.h

**Prototype** UInt16 LmGetNumLocales (void)

**Parameters** None.

**Result** Returns the number of locales that the locale system resource defines.

**Comments** Use this function to obtain the range of possible values that you can pass as an index to [LmGetLocaleSetting](#). If LmGetNumLocales returns 3, then LmGetLocaleSetting accepts indexes in the range of 0 to 2.

This function returns only the number of locales for which the ROM has locale information. It does not return the number of locales that could possibly be defined. For example, the system resource currently contains no locale whose language is 1Hebrew and country is cIsrael, even though that is a valid locale.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call [LmGlueGetNumLocales](#). For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

## LmLocaleToIndex

**Purpose** Convert an [LmLocaleType](#) to an index.

**Declared In** LocaleMgr.h

**Prototype** Err LmLocaleToIndex (const LmLocaleType \*iLocale,  
                  UInt16 \*oLocaleIndex)

**Parameters** -> iLocale         The locale to convert.

<- oLocaleIndex   The index of iLocale upon return.

**Result** Returns errNone upon success or lmErrUnknownLocale if the locale could not be found.

**Comments** You can use this function to obtain a valid index to pass to [LmGetLocaleSetting](#). For example, you might use the Overlay Manager routine [OmGetSystemLocale](#) to return the locale used on the current system and then pass that locale to this function to obtain its index.

```
LmLocaleType locale;
Char oValue [kMaxCurrencySymbolLen+1];
UInt16 index;

OmGetSystemLocale(&locale);
LmLocaleToIndex(&locale, &index);
LmGetLocaleSetting(index,
    lmChoiceCurrencySymbol, oValue,
    sizeof(oValue));
```

The LmLocaleType that is passed in iLocale can use the constants lmAnyCountry or lmAnyLanguage as wildcards. For example, if the country is lmAnyCountry, LmLocaleToIndex returns the index of the first locale that matches the language.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call

## **Locale Manager**

### *Locale Manager Functions*

---

`LmGlueLocaleToIndex`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

# Memory Manager

---

This chapter provides reference information for the memory manager. The memory manager API is declared in the header file `MemoryMgr.h`.

For more information on the memory manager, see the chapter “[Memory](#)” in the *Palm OS Programmer’s Companion*, vol. I.

## Memory Manager Functions

### MemCardInfo

**Purpose** Return information about a memory card.

**Declared In** `MemoryMgr.h`

**Prototype** `Err MemCardInfo (UInt16 cardNo, Char* cardNameP,  
Char* manufNameP, UInt16* versionP,  
UInt32* crDateP, UInt32* romSizeP,  
UInt32* ramSizeP, UInt32* freeBytesP)`

<b>Parameters</b>	<code>cardNo</code>	Card number.
	<code>cardNameP</code>	Pointer to character array (32 bytes), or 0.
	<code>manufNameP</code>	Pointer to character array (32 bytes), or 0.
	<code>versionP</code>	Pointer to version variable, or 0.
	<code>crDateP</code>	Pointer to creation date variable, or 0.
	<code>romSizeP</code>	Pointer to ROM size variable, or 0.
	<code>ramSizeP</code>	Pointer to RAM size variable, or 0.

## Memory Manager

### Memory Manager Functions

---

freeBytesP      Pointer to free byte-count variable, or 0.

**Result**      Returns 0 if no error.

**Comments**      Pass 0 for those variables that you don't want returned.

## MemCmp

**Purpose**      Compare two blocks of memory.

---

**NOTE:**      Blocks are compared as unsigned bytes.

---

**Declared In**      MemoryMgr.h

**Prototype**      Int16 MemCmp (const void\* s1, const void\* s2,  
                      Int32 numBytes)

**Parameters**      s1, s2      Pointers to block of memory.  
                      numBytes      Number of bytes to compare.

**Result**      Zero if they match, non-zero if not:

- + if s1 > s2
- if s1 < s2

**Compatibility**      Implemented only if [2.0 New Feature Set](#) is present.

MemCmp can be used to test the equality of blocks in memory on all versions that support MemCmp; however, testing the sort ordering of blocks in memory works reliably only on Palm OS® versions 3.5 and higher. On versions earlier than 3.2, MemCmp always returns a positive value if the blocks are unequal. On versions 3.2 and 3.3, MemCmp reliably returns positive to indicate s1 > s2 and negative to indicate s1 < s2 **only** if the characters that differ are less than 128 apart. If the difference is greater than that, MemCmp may return positive when it should return negative and vice versa.

## MemDebugMode

**Purpose** Return the current debugging mode of the memory manager.

**Declared In** MemoryMgr.h

**Prototype** UInt16 MemDebugMode (void)

**Parameters** No parameters.

**Result** Returns debug flags as described for [MemSetDebugMode](#).

## MemHandleCardNo

**Purpose** Return the card number a chunk resides in.

**Declared In** MemoryMgr.h

**Prototype** UInt16 MemHandleCardNo (MemHandle h)

**Parameters** -> h                          Chunk handle.

**Result** Returns the card number.

**Comments** Call this routine to retrieve the card number (0 or 1) a movable chunk resides on.

**See Also** [MemPtrCardNo](#)

## **Memory Manager**

### *Memory Manager Functions*

---

## **MemHandleDataStorage**

<b>Purpose</b>	Return true if the given handle is part of a data storage heap. If not, it's a handle in the dynamic heap.
<b>Declared In</b>	MemoryMgr.h
<b>Prototype</b>	Boolean MemHandleDataStorage (MemHandle h)
<b>Parameters</b>	-> h Chunk handle.
<b>Result</b>	Returns true if the handle is part of a data storage heap.
<b>Comments</b>	Called by Fields package routines to determine if they need to worry about data storage write-protection when editing a text field.
<b>See Also</b>	<a href="#">MemPtrDataStorage</a>

## **MemHandleFree**

<b>Purpose</b>	Dispose of a movable chunk.
<b>Declared In</b>	MemoryMgr.h
<b>Prototype</b>	Err MemHandleFree (MemHandle h)
<b>Parameters</b>	-> h Chunk handle.
<b>Result</b>	Returns 0 if no error, or memErrInvalidParam if an error occurs.
<b>Comments</b>	Call this routine to dispose of a movable chunk.
<b>See Also</b>	<a href="#">MemHandleNew</a>

## MemHandleHeapID

**Purpose** Return the heap ID of a chunk.

**Declared In** MemoryMgr.h

**Prototype** UInt16 MemHandleHeapID (MemHandle h)

**Parameters** -> h                          Chunk handle.

**Result** Returns the heap ID of a chunk.

**Comments** Call this routine to get the heap ID of the heap a chunk resides in.

**See Also** [MemPtrHeapID](#)

## MemHandleLock

**Purpose** Lock a chunk and obtain a pointer to the chunk's data.

**Declared In** MemoryMgr.h

**Prototype** MemPtr MemHandleLock (MemHandle h)

**Parameters** -> h                          Chunk handle.

**Result** Returns a pointer to the chunk.

**Comments** Call this routine to lock a chunk and obtain a pointer to the chunk.  
MemHandleLock and MemHandleUnlock should be used in pairs.

**See Also** [MemHandleNew](#), [MemHandleUnlock](#)

## Memory Manager

### Memory Manager Functions

---

## MemHandleNew

<b>Purpose</b>	Allocate a new movable chunk in the dynamic heap and returns a handle to it.
<b>Declared In</b>	MemoryMgr.h
<b>Prototype</b>	MemHandle MemHandleNew (UInt32 size)
<b>Parameters</b>	-> size                          The desired size of the chunk.
<b>Result</b>	Returns a handle to the new chunk, or 0 if unsuccessful.
<b>Comments</b>	Use this call to allocate dynamic memory. Before you can write data to the memory chunk that MemHandleNew allocates, you must call <a href="#">MemHandleLock</a> to lock the chunk and get a pointer to it.
<b>See Also</b>	<a href="#">MemPtrFree</a> , <a href="#">MemPtrNew</a> , <a href="#">MemHandleFree</a> , <a href="#">MemHandleLock</a>

## MemHandleResize

<b>Purpose</b>	Resize a chunk.
<b>Declared In</b>	MemoryMgr.h
<b>Prototype</b>	Err MemHandleResize (MemHandle h, UInt32 newSize)
<b>Parameters</b>	-> h                                  Chunk handle. -> newSize                            The new desired size.
<b>Result</b>	0                                      No error.  <code>memErrInvalidParam</code> Invalid parameter passed.  <code>memErrNotEnoughSpace</code> Not enough free space in heap to grow chunk.

`memErrChunkLocked`

Can't grow chunk because it's locked.

<b>Comments</b>	Call this routine to resize a chunk. This routine is always successful when shrinking the size of a chunk, even if the chunk is locked. When growing a chunk, it first attempts to grab free space immediately following the chunk so that the chunk does not have to move. If the chunk has to move to another free area of the heap to grow, it must be movable and have a lock count of 0.  On devices running version 2.0 or earlier of Palm OS, the <code>MemHandleResize</code> function tries to resize the chunk only within the same heap, whereas <a href="#">DmResizeRecord</a> will look for space in other data heaps if it can't find enough space in the original heap.
<b>See Also</b>	<a href="#">MemHandleNew</a> , <a href="#">MemHandleSize</a>

## MemHandleSetOwner

<b>Purpose</b>	Set the owner ID of a chunk.				
<b>Declared In</b>	<code>MemoryMgr.h</code>				
<b>Prototype</b>	<code>Err MemHandleSetOwner (MemHandle h, UInt16 owner)</code>				
<b>Parameters</b>	<table><tr><td><code>-&gt; h</code></td><td>Chunk handle.</td></tr><tr><td><code>-&gt; owner</code></td><td>New owner ID of the chunk. Specify 0 to set the owner to the operating system.</td></tr></table>	<code>-&gt; h</code>	Chunk handle.	<code>-&gt; owner</code>	New owner ID of the chunk. Specify 0 to set the owner to the operating system.
<code>-&gt; h</code>	Chunk handle.				
<code>-&gt; owner</code>	New owner ID of the chunk. Specify 0 to set the owner to the operating system.				
<b>Result</b>	Returns 0 if no error, or <code>memErrInvalidParam</code> if an error occurs.				
<b>Comments</b>	When you allocate a parameter block to pass to <a href="#">SysUIAppSwitch</a> or <a href="#">SysAppLaunch</a> , you must call <a href="#">MemPtrSetOwner</a> to grant ownership of the parameter block chunk to the OS (your application is originally set as the owner). If the parameter block structure references any chunks by handle, you also must call <code>MemHandleSetOwner</code> to grant ownership of those blocks to the OS. If you don't change the ownership of these chunks, they will get				

## **Memory Manager**

### *Memory Manager Functions*

---

freed before the application you're launching has a chance to use them.

## **MemHandleSize**

**Purpose** Return the requested size of a chunk.

**Declared In** MemoryMgr.h

**Prototype** UInt32 MemHandleSize (MemHandle h)

**Parameters** -> h                          Chunk handle.

**Result** Returns the requested size of the chunk.

**Comments** Call this routine to get the size originally requested for a chunk.

**See Also** [MemHandleResize](#)

## **MemHandleToLocalID**

**Purpose** Convert a handle into a local chunk ID which is card relative.

**Declared In** MemoryMgr.h

**Prototype** LocalID MemHandleToLocalID (MemHandle h)

**Parameters** -> h                          Chunk handle.

**Result** Returns local ID, or NULL (0) if unsuccessful.

**Comments** Call this routine to convert a chunk handle to a local ID.

**See Also** [MemLocalIDToGlobal](#), [MemLocalIDToLockedPtr](#)

## MemHandleUnlock

**Purpose** Unlock a chunk given a chunk handle.

**Declared In** MemoryMgr.h

**Prototype** Err MemHandleUnlock (MemHandle h)

**Parameters** -> h The chunk handle.

**Result** 0 No error.

memErrInvalidParam  
Invalid parameter passed.

**Comments** Call this routine to decrement the lock count for a chunk.

MemHandleLock and MemHandleUnlock should be used in pairs.

**See Also** [MemHandleLock](#)

## MemHeapCheck

**Purpose** Check validity of a given heap.

**Declared In** MemoryMgr.h

**Prototype** Err MemHeapCheck (UInt16 heapID)

**Parameters** heapID ID of heap to check.

**Result** Returns 0 if no error.

**See Also** [MemDebugMode](#), [MemSetDebugMode](#)

## **Memory Manager**

### *Memory Manager Functions*

---

## **MemHeapCompact**

**Purpose** Compact a heap.

**Declared In** MemoryMgr.h

**Prototype** Err MemHeapCompact (UInt16 heapID)

**Parameters** -> heapID ID of the heap to compact.

**Result** Always returns 0.

**Comments** Most applications never need to call this function explicitly. The system software calls this function at various times; for example, during memory allocation (if sufficient free space is not available) and during system reboot.

Call this routine to compact a heap and merge all free space. This routine attempts to move all movable chunks to the start of the heap and merge all free space in the center of the heap.

## **MemHeapDynamic**

**Purpose** Return true if the given heap is a dynamic heap.

**Declared In** MemoryMgr.h

**Prototype** Boolean MemHeapDynamic (UInt16 heapID)

**Parameters** heapID ID of the heap to be tested.

**Result** Returns true if dynamic, false if not.

**Comments** Dynamic heaps are used for volatile storage, application stacks, globals, and dynamically allocated memory.

**NOTE:** In Palm OS 3.5, the dynamic heap is sized based on the amount of memory available, and is generally larger than before.

---

**See Also** [MemNumHeaps](#), [MemHeapID](#)

## MemHeapFlags

**Purpose** Return the heap flags for a heap.

**Declared In** MemoryMgr.h

**Prototype** UInt16 MemHeapFlags (UInt16 heapID)

**Parameters** -> heapID ID of heap.

**Result** Returns the heap flags.

**Comments** Call this routine to retrieve the heap flags for a heap. The flags can be examined to determine if the heap is ROM based or not. ROM-based heaps have the memHeapFlagReadOnly bit set (the memHeapFlagReadOnly mask has a value of 0x0001).

**See Also** [MemNumHeaps](#), [MemHeapID](#)

## MemHeapFreeBytes

**Purpose** Return the total number of free bytes in a heap and the size of the largest free chunk in the heap.

**Declared In** MemoryMgr.h

**Prototype** Err MemHeapFreeBytes (UInt16 heapID,  
                  UInt32\* freeP, UInt32\* maxP)

**Parameters** -> heapID ID of heap.

## Memory Manager

### Memory Manager Functions

---

<-> freeP	Pointer to a variable of type UInt32 for free bytes.
<-> maxP	Pointer to a variable of type UInt32 for max free chunk size. Do not pass NULL for this argument.

**Result** Always returns 0.

**Comments** This routine doesn't compact the heap but may be used to determine in advance whether an allocation request will succeed. Before allocating memory, call this function and test the value returned in maxP to determine whether enough free space to fulfill your allocation request exists. If not, you may make more space available by calling the [MemHeapCompact](#) function. Note that both [MemPtrNew](#) and [MemHandleNew](#) automatically compact the heap if necessary.

**See Also** [MemHeapSize](#), [MemHeapID](#), [MemHeapCompact](#)

## MemHeapID

**Purpose** Return the heap ID for a heap, given its index and the card number.

**Declared In** MemoryMgr.h

**Prototype** UInt16 MemHeapID (UInt16 cardNo, UInt16 heapIndex)

**Parameters**

-> cardNo	The card number, either 0 or 1.
-> heapIndex	The heap index, anywhere from 0 to <a href="#">MemNumHeaps</a> - 1.

**Result** Returns the heap ID.

**Comments** Call this routine to retrieve the heap ID of a heap, given the heap index and the card number. A heap ID must be used to obtain

information on a heap such as its size, free bytes, etc., and is also passed to any routines which manipulate heaps.

**See Also** [MemNumHeaps](#)

## MemHeapScramble

**Purpose** Scramble the specified heap.

**Declared In** MemoryMgr.h

**Prototype** Err MemHeapScramble (UInt16 heapID)

**Parameters** heapID ID of heap to scramble.

**Comments** The system attempts to move each movable chunk.  
Useful for debugging.

**Result** Always returns 0.

**See Also** [MemDebugMode](#), [MemSetDebugMode](#)

## MemHeapSize

**Purpose** Return the total size of a heap including the heap header.

**Declared In** MemoryMgr.h

**Prototype** UInt32 MemHeapSize (UInt16 heapID)

**Parameters** -> heapID ID of heap.

**Result** Returns the total size of the heap.

**See Also** [MemHeapFreeBytes](#), [MemHeapID](#)

## **Memory Manager**

### *Memory Manager Functions*

---

## **MemLocalIDKind**

**Purpose** Return whether or not a local ID references a handle or a pointer.

**Declared In** MemoryMgr.h

**Prototype** LocalIDKind MemLocalIDKind (LocalID local)

**Parameters** -> local                Local ID to query

**Result** Returns LocalIDKind, or a memIDHandle or memIDPtr (see MemoryMgr.h).

**Comments** This routine determines if the given local ID is to a nonmovable (memIDPtr) or movable (memIDHandle) chunk.

## **MemLocalIDToGlobal**

**Purpose** Convert a local ID, which is card relative, into a global pointer in the designated card.

**Declared In** MemoryMgr.h

**Prototype** MemPtr MemLocalIDToGlobal (LocalID local,  
                  UInt16 cardNo)

**Parameters** -> local                The local ID to convert.

-> cardNo                Memory card the chunk resides in.

**Result** Returns pointer or handle to chunk.

**See Also** [MemLocalIDKind](#), [MemLocalIDToLockedPtr](#)

## MemLocalIDToLockedPtr

**Purpose** Return a pointer to a chunk given its local ID and card number.  
If the local ID references a movable chunk handle, this routine automatically locks the chunk before returning.

**Declared In** MemoryMgr.h

**Prototype** MemPtr MemLocalIDToLockedPtr (LocalID local,  
UInt16 cardNo)

**Parameters** local Local chunk ID.  
cardNo Card number.

**Result** Returns pointer to chunk, or 0 if an error occurs.

**See Also** [MemLocalIDToGlobal](#), [MemLocalIDToPtr](#), [MemLocalIDKind](#),  
[MemPtrToLocalID](#), [MemHandleToLocalID](#)

## MemLocalIDToPtr

**Purpose** Return pointer to chunk, given the local ID and card number.

**Declared In** MemoryMgr.h

**Prototype** MemPtr MemLocalIDToPtr (LocalID local,  
UInt16 cardNo)

**Parameters** -> local Local ID to query.  
-> cardNo Card number the chunk resides in.

**Result** Returns a pointer to the chunk, or 0 if error.

**Comments** If the local ID references a movable chunk and that chunk is **not** locked, this function returns 0 to indicate an error.

**See Also** [MemLocalIDToGlobal](#), [MemLocalIDToLockedPtr](#)

## **Memory Manager**

### *Memory Manager Functions*

---

## **MemMove**

**Purpose** Move a range of memory to another range in the dynamic heap.

**Declared In** MemoryMgr.h

**Prototype** Err MemMove (void\* dstP, const void\* sP,  
Int32 numBytes)

**Parameters** dstP Pointer to destination.  
sP Pointer to source.  
numBytes Number of bytes to move.

**Result** Always returns 0.

**Comments** Handles overlapping ranges.

For operations where the destination is in a data heap, see [DmSet](#), [DmWrite](#), and related functions.

## **MemNumCards**

**Purpose** Return the number of memory card slots in the system. Not all slots need to be populated.

**Declared In** MemoryMgr.h

**Prototype** UInt16 MemNumCards (void)

**Parameters** None.

**Result** Returns number of slots in the system.

## MemNumHeaps

**Purpose** Return the number of heaps available on a particular card.

**Declared In** MemoryMgr.h

**Prototype** UInt16 MemNumHeaps (UInt16 cardNo)

**Parameters** -> cardNo The card number; either 0 or 1.

**Result** Number of heaps available, including ROM- and RAM-based heaps.

**Comments** Call this routine to retrieve the total number of heaps on a memory card. The information can be obtained by calling [MemHeapSize](#), [MemHeapFreeBytes](#), and [MemHeapFlags](#) on each heap using its heap ID. The heap ID is obtained by calling [MemHeapID](#) with the card number and the heap index, which can be any value from 0 to MemNumHeaps.

## MemNumRAMHeaps

**Purpose** Return the number of RAM heaps in the given card.

**Declared In** MemoryMgr.h

**Prototype** UInt16 MemNumRAMHeaps (UInt16 cardNo)

**Parameters** cardNo The card number.

**Result** Returns the number of RAM heaps.

**See Also** [MemNumCards](#)

## **Memory Manager**

### *Memory Manager Functions*

---

## **MemPtrCardNo**

**Purpose** Return the card number (0 or 1) a nonmovable chunk resides on.

**Declared In** MemoryMgr.h

**Prototype** UInt16 MemPtrCardNo (MemPtr p)

**Parameters** -> p                    Pointer to the chunk.

**Result** Returns the card number.

**See Also** [MemHandleCardNo](#)

## **MemPtrDataStorage**

**Purpose** Return true if the given pointer is part of a data storage heap; if not, it is a pointer in the dynamic heap.

**Declared In** MemoryMgr.h

**Prototype** Boolean MemPtrDataStorage (MemPtr p)

**Parameters** p                    Pointer to a chunk.

**Result** Returns true if the chunk is part of a data storage heap.

**Comments** Called by Fields package to determine if it needs to worry about data storage write-protection when editing a text field.

**See Also** [MemHeapDynamic](#)

## MemPtrFree

**Purpose** Macro to dispose of a chunk.

**Declared In** MemoryMgr.h

**Prototype** Err MemPtrFree (MemPtr p)

**Parameters** -> p                    Pointer to a chunk.

**Result** 0                    If no error or memErrInvalidParam (invalid parameter).

**Comments** Call this routine to dispose of a nonmovable chunk.

## MemPtrHeapID

**Purpose** Return the heap ID of a chunk.

**Declared In** MemoryMgr.h

**Prototype** UInt16 MemPtrHeapID (MemPtr p)

**Parameters** -> p                    Pointer to the chunk.

**Result** Returns the heap ID of a chunk.

**Comments** Call this routine to get the heap ID of the heap a chunk resides in.

## Memory Manager

### *Memory Manager Functions*

---

## MemPtrNew

<b>Purpose</b>	Allocate a new nonmovable chunk in the dynamic heap.
<b>Declared In</b>	MemoryMgr.h
<b>Prototype</b>	MemPtr MemPtrNew (UInt32 size)
<b>Parameters</b>	-> size                         The desired size of the chunk.
<b>Result</b>	Returns pointer to the new chunk, or 0 if unsuccessful.
<b>Comments</b>	This routine allocates a nonmovable chunk in the dynamic heap and returns a pointer to the chunk. Applications can use it when allocating dynamic memory. Note that chunks range in size from 1 byte to slightly less than 64KB; you cannot allocate a single chunk larger than this.  In Palm OS 3.5, the dynamic heap is sized based on the amount of memory available, and is generally larger than before.

---

**NOTE:** You cannot allocate a zero-size reference block.

---

## MemPtrRecoverHandle

<b>Purpose</b>	Recover the handle of a movable chunk, given a pointer to its data.
<b>Declared In</b>	MemoryMgr.h
<b>Prototype</b>	MemHandle MemPtrRecoverHandle (MemPtr p)
<b>Parameters</b>	-> p                             Pointer to the chunk.
<b>Result</b>	Returns the handle of the chunk, or 0 if unsuccessful.
<b>Comments</b>	Don't call this function for pointers in ROM or nonmovable data chunks.

## MemPtrResize

**Purpose** Resize a chunk.

**Declared In** MemoryMgr.h

**Prototype** Err MemPtrResize (MemPtr p, UInt32 newSize)

**Parameters** -> p Pointer to the chunk.  
-> newSize The new desired size.

**Result** Returns 0 if no error, or memErrNotEnoughSpace  
memErrInvalidParam, or memErrChunkLocked if an error occurs.

**Comments** Call this routine to resize a locked chunk. This routine is always successful when shrinking the size of a chunk. When growing a chunk, it attempts to use free space immediately following the chunk.

**See Also** [MemPtrSize](#), [MemHandleResize](#)

## MemPtrSetOwner

**Purpose** Set the owner ID of a chunk.

**Declared In** MemoryMgr.h

**Prototype** Err MemPtrSetOwner (MemPtr p, UInt16 owner)

**Parameters** -> p Pointer to the chunk.  
-> owner New owner ID of the chunk. Specify 0 to set the owner to the operating system.

**Result** Returns 0 if no error, or memErrInvalidParam if an error occurs.

## Memory Manager

### Memory Manager Functions

---

**Comments** When you allocate a parameter block to pass to [SysUIAppSwitch](#) or [SysAppLaunch](#), you must call `MemPtrSetOwner` or [MemHandleSetOwner](#) to grant ownership of the parameter block chunk, and any other chunks referenced in it, to the OS (your application is originally set as the owner). If you don't change the ownership of the parameter block, it will get freed before the application you're launching has a chance to use it.

## MemPtrSize

**Purpose** Return the size of a chunk.

**Declared In** MemoryMgr.h

**Prototype** `UInt32 MemPtrSize (MemPtr p)`

**Parameters** `-> p` Pointer to the chunk.

**Result** The requested size of the chunk.

**Comments** Call this routine to get the original requested size of a chunk.

## MemPtrToLocalID

**Purpose** Convert a pointer into a card-relative local chunk ID.

**Declared In** MemoryMgr.h

**Prototype** `LocalID MemPtrToLocalID (MemPtr p)`

**Parameters** `-> p` Pointer to a chunk.

**Result** Returns the local ID of the chunk.

**Comments** Call this routine to convert a chunk pointer to a local ID.

**See Also** [MemLocalIDToPtr](#)

## MemPtrUnlock

**Purpose** Unlock a chunk, given a pointer to the chunk.

**Declared In** MemoryMgr.h

**Prototype** Err MemPtrUnlock (MemPtr p)

**Parameters** p                            Pointer to a chunk.

**Result** 0 if no error, or memErrInvalidParam if an error occurs.

**Comments** A chunk must **not** be unlocked more times than it was locked.

**See Also** [MemHandleLock](#)

## MemSet

**Purpose** Set a memory range in a dynamic heap to a specific value.

**Declared In** MemoryMgr.h

**Prototype** Err MemSet (void\* dstP, Int32 numBytes,  
                  UInt8 value)

## Memory Manager

### Memory Manager Functions

---

<b>Parameters</b>	dstP	Pointer to the destination.
	numBytes	Number of bytes to set.
	value	Value to set.

**Result** Always returns 0.

**Comments** For operations where the destination is in a data heap, see [DmSet](#), [DmWrite](#), and related functions.

## MemSetDebugMode

**Purpose** Set the debugging mode of the memory manager.

**Declared In** MemoryMgr.h

**Prototype** Err MemSetDebugMode (UInt16 flags)

<b>Parameters</b>	flags	Debug flags.
-------------------	-------	--------------

**Comments** Use the logical OR operator (|) to provide any combination of one, more, or none of the following flags:

memDebugModeCheckOnChange  
memDebugModeCheckOnAll  
memDebugModeScrambleOnChange  
memDebugModeScrambleOnAll  
memDebugModeFillFree  
memDebugModeAllHeaps  
memDebugModeRecordMinDynHeapFree

**Result** Returns 0 if no error, or -1 if an error occurs.

## MemStoreInfo

**Purpose** Return information on either the RAM store or the ROM store for a memory card.

**Declared In** MemoryMgr.h

**Prototype**

```
Err MemStoreInfo (UInt16 cardNo,
                  UInt16 storeNumber, UInt16* versionP,
                  UInt16* flagsP, Char* nameP, UInt32* crDateP,
                  UInt32* bckUpDateP, UInt32* heapListOffsetP,
                  UInt32* initCodeOffset1P,
                  UInt32* initCodeOffset2P, LocalID* databaseDirIDP)
```

**Parameters**

-> cardNo	Card number, either 0 or 1.
-> storeNumber	Store number; 0 for ROM, 1 for RAM.
<-> versionP	Pointer to version variable, or 0.
<-> flagsP	Pointer to flags variable, or 0.
<-> nameP	Pointer to character array (32 bytes), or 0.
<-> crDateP	Pointer to creation date variable, or 0.
<-> bckUpDateP	Pointer to backup date variable, or 0.
<-> heapListOffsetP	Pointer to heapListOffset variable, or 0.
<-> initCodeOffset1P	Pointer to initCodeOffset1 variable, or 0.
<-> initCodeOffset2P	Pointer to initCodeOffset2 variable, or 0.
<-> databaseDirIDP	Pointer to database directory chunk ID variable, or 0.

**Result** Returns 0 if no error, or memErrCardNotPresent, memErrRAMOnlyCard, or memErrInvalidStoreHeader if an error occurs.

## **Memory Manager**

### *Memory Manager Functions*

---

**Comments** Call this routine to retrieve any or all information on either the RAM store or the ROM store for a card. Pass 0 for variables that you don't wish returned.

# Notification Manager

---

This chapter describes the Notification Manager API as declared in the header file `NotifyMgr.h`. It discusses the following topics:

- [Notification Constants](#)
- [Notification Functions](#)
- [Application-Defined Functions](#)

The chapter “[Notifications](#)” in this book lists the possible notifications and describes the data sent with each. Also see the section “[Notifications](#)” on page 30 in the “[Application Startup and Stop](#)” chapter of the *Palm OS Programmer’s Companion*, vol. I for a description of how to use notifications.

## Notification Constants

### Miscellaneous Constants

The following miscellaneous constants are used in the Notification Manager. For other notification constants, see the “[Notifications](#)” chapter in this book.

Constant	Value	Description
<code>sysNotifyBroadcasterCode</code>	'psys'	The value passed as the creator ID of the broadcaster for notifications broadcast by the system.
<code>sysNotifyDefaultQueueSize</code>	30	The maximum number of nested broadcasts allowed.
<code>sysNotifyNoDatabaseID</code>	0xFFFFFFFF	The database local ID used by the system when it registers for notifications.

## **Notification Manager**

### *Notification Functions*

---

<b>Constant</b>	<b>Value</b>	<b>Description</b>
sysNotifyNormalPriority	0	Typical priority value used when registering for notifications.
sysNotifyVersionNum	1	Current Notification Manager version. This number is stored in the system feature <code>sysFtrNumNotifyMgrVersion</code> .

## **Notification Functions**

### **SysNotifyBroadcast**

**Purpose** Synchronously send a notification to all applications registered for it.

**Declared In** NotifyMgr.h

**Prototype** Err SysNotifyBroadcast  
(SysNotifyParamType \*notify)

**Parameters** <-> notify      Identifies the notification to be broadcast. See [SysNotifyParamType](#).

**Result** Returns one of the following error codes:

errNone      No error.

sysNotifyErrBroadcastBusy      The broadcast stack limit has already been reached.

sysErrParamErr      The background thread is broadcasting the notification and the notify parameter is NULL.

sysNotifyErrNoStackSpace      There is not enough space on the stack for the notification.

**Comments** When you call this function, the notification you specify is broadcast to all applications, shared libraries, and other code resources that have registered to receive that notification. The broadcast is performed synchronously, meaning that the system broadcasts the notification immediately and waits for each notification client to perform its notification handler and return before the `SysNotifyBroadcast` call returns. This notification occurs in priority order.

The system allows nested notifications; that is, the recipient of a notification might broadcast a new notification, whose recipient might broadcast another new notification and so on. The constant `sysNotifyDefaultQueueSize` specifies how many levels of nested notification are allowed. If you reach this limit, the error `sysNotifyErrBroadcastBusy` is returned and your notification is not broadcast. To avoid reaching the limit, use [SysNotifyBroadcastDeferred](#) instead of [SysNotifyBroadcast](#) in your notification handlers. This ensures that the notification will not be broadcast until the top of the event loop.

---

**WARNING!** Do not call `SysNotifyBroadcast` from code that might be called from a background task (such as a trap patch) with the memory semaphore reserved. Use `SysNotifyBroadcastDeferred` instead.

---

**Compatibility** Implemented only if [Notification Feature Set](#) is present.

## Notification Manager

### Notification Functions

---

## SysNotifyBroadcastDeferred

**Purpose** Enqueue a notification for later broadcast.

**Declared In** NotifyMgr.h

**Prototype** Err SysNotifyBroadcastDeferred  
(SysNotifyParamType \*notify, Int16 paramSize)

**Parameters** <-> notify The notification to enqueue. See  
[SysNotifyParamType](#).

-> paramSize Size of the data pointed to by the field  
notify->notifyDetailsP.

**Result** Returns one of the following error codes:

errNone No error.

memErrNotEnoughSpace There is not enough memory to allocate a new  
notification entry in the queue.

sysErrParamErr paramSize is a negative number.

sysNotifyErrQueueFull The queue has reached its maximum number of  
entries.

**Comments** This function is similar to [SysNotifyBroadcast](#) except that the  
broadcast does not take place until the top of the event loop  
(specifically, the next time [EvtGetEvent](#) is called). The system  
copies the notify structure to a new memory chunk, which is  
disposed of upon completion of the broadcast. (The paramSize  
value is used when copying the notifyDetailsP portion of the  
notify structure.)

**Compatibility** Implemented only if [Notification Feature Set](#) is present.

## SysNotifyBroadcastFromInterrupt

<b>Purpose</b>	Allows interrupt handlers to enqueue a notification for later broadcast.						
<b>Declared In</b>	NotifyMgr.h						
<b>Prototype</b>	<pre>Err SysNotifyBroadcastFromInterrupt (UInt32 notifyType, UInt32 broadcaster, void *notifyDetailsP)</pre>						
<b>Parameters</b>	<table><tr><td>-&gt; notifyType</td><td>The type of event that occurred. See the chapter <a href="#">Notifications</a> for a complete list of the notifications that Palm OS® broadcasts.</td></tr><tr><td>-&gt; broadcaster</td><td>The creator ID of the device or application that broadcast the notification.</td></tr><tr><td>-&gt; notifyDetailsP</td><td>Pointer to data specific to this notification. See the <a href="#">Notifications</a> chapter for the specific instances where this parameter is used.</td></tr></table>	-> notifyType	The type of event that occurred. See the chapter <a href="#">Notifications</a> for a complete list of the notifications that Palm OS® broadcasts.	-> broadcaster	The creator ID of the device or application that broadcast the notification.	-> notifyDetailsP	Pointer to data specific to this notification. See the <a href="#">Notifications</a> chapter for the specific instances where this parameter is used.
-> notifyType	The type of event that occurred. See the chapter <a href="#">Notifications</a> for a complete list of the notifications that Palm OS® broadcasts.						
-> broadcaster	The creator ID of the device or application that broadcast the notification.						
-> notifyDetailsP	Pointer to data specific to this notification. See the <a href="#">Notifications</a> chapter for the specific instances where this parameter is used.						
<b>Result</b>	Returns one of the following error codes:  <table><tr><td>errNone</td><td>No error.</td></tr><tr><td>sysNotifyErrQueueFull</td><td>The queue has reached its maximum number of entries.</td></tr></table>	errNone	No error.	sysNotifyErrQueueFull	The queue has reached its maximum number of entries.		
errNone	No error.						
sysNotifyErrQueueFull	The queue has reached its maximum number of entries.						
<b>Comments</b>	<p>Like <a href="#">SysNotifyBroadcastDeferred</a>, this function enqueues a notification to be broadcast at the top of the event loop (specifically, the next time <a href="#">EvtGetEvent</a> is called). It differs from <a href="#">SysNotifyBroadcastDeferred</a> in that it is interrupt-safe and intended to be called from interrupt handlers.</p> <p>This function is intended to be used by device drivers and other low-level software to generate a notification about a hardware change. For example, the Expansion Manager uses <a href="#">SysNotifyBroadcastFromInterrupt</a> to broadcast <a href="#">sysNotifyCardInsertedEvent</a> and <a href="#">sysNotifyCardEjectedEvent</a>.</p>						

## Notification Manager

### Notification Functions

---

[sysNotifyCardRemovedEvent](#) when a card is inserted into or removed from the expansion slot.

SysNotifyBroadcastFromInterrupt is not intended to be used by general third party applications. Patching SysNotifyBroadcastFromInterrupt will cause the system to hang.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## SysNotifyRegister

**Purpose** Register to receive a notification.

**Declared In** NotifyMgr.h

**Prototype** Err SysNotifyRegister (UInt16 cardNo,  
LocalID dbID, UInt32 notifyType,  
SysNotifyProcPtr callbackP, Int8 priority,  
void \*userDataP)

<b>Parameters</b>	-> cardNo	Number of the storage card on which the application or code resource resides.
	-> dbID	Local ID of the application or code resource.
	-> notifyType	The notification that the application wants to receive. See the chapter <a href="#">Notifications</a> .
	-> callbackP	Set to NULL to receive the notification as an application launch code. If your code does not have a <a href="#">PilotMain</a> function (for example, if it is a shared library), pass a pointer to a function that should be called when the notification is broadcast. See <a href="#">SysNotifyProcPtr</a> .

-> priority	The priority with which the application should receive the event. Most applications and other code resources should always use <code>sysNotifyNormalPriority</code> . In rare circumstances, you may need to ensure that your code is notified toward the beginning or toward the end of the notification sequence. If so, be sure to leave some space so that your code won't collide with the system's handling of notifications or with another application's handling of notifications. In general, Palm™ recommends using a value whose least significant bits are 0 (such as 32, 64, 96, and so on). The smaller the priority, the earlier your code is notified.
-> userDataP	Caller-defined data to pass to the notification handler.

<b>Result</b>	Returns one of the following error codes:
<code>errNone</code>	No error.
<code>sysErrParamErr</code>	The database ID is NULL.
<code>sysNotifyErrDuplicateEntry</code>	This application is already registered to receive this notification.

<b>Comments</b>	Call this function when your code should receive a notification that a specific event has occurred or is about to occur. See the <a href="#">Notifications</a> chapter for a list of the possible notifications. Once you register for a notification, you remain registered to receive it until a system reset occurs or until you explicitly unregister using <a href="#"><u>SysNotifyUnregister</u></a> .  If you're writing an application, you should pass NULL as the <code>callbackP</code> parameter. In this case, the system notifies your application by sending it the <a href="#"><u>sysAppLaunchCmdNotify</u></a> launch code. This launch code's parameter block points to a <a href="#"><u>SysNotifyParamType</u></a> structure containing details about the notification.
-----------------	--

## Notification Manager

### Notification Functions

---

If your code is not in an application, for example, it is a shared library or a separately loaded code resource, then receiving a launch code is not possible. In this case, pass a pointer to a callback function in `callbackP`. This callback should follow the prototype shown in [SysNotifyProcPtr](#). Note that you should always supply a card number and database ID to `SysNotifyRegister`, even if you specify a callback function.

---

**IMPORTANT:** Because the `callbackP` pointer is used to directly call the function, the pointer must remain valid from the time `SysNotifyRegister` is called to the time the notification is broadcast. If the function is in a shared library, you must keep the library open. If the function is in a separately loaded code resource, the resource must remain locked while registered for the notification. When you close a library or unlock a resource, you must first unregister for any notifications. If you don't, the system will crash when the notification is broadcast.

---

Whether the notification handler is responding to `sysAppLaunchCmdNotify` or uses the callback function, the notification handler may perform any processing necessary. As with most launch codes, it's not possible to access global variables. If the handler needs access to any particular value to respond to the notification, pass a pointer to that value in the `userDataP` parameter. The system passes this pointer back to your application or callback function in the launch code's parameter block.

The notification handler may unregister for this notification or register for other notifications. It may also broadcast another notifications; however, it's recommended that you use [SysNotifyBroadcastDeferred](#) to do this so as not to overflow the broadcast stack.

You may display a user interface in your notification handler; however, you should be careful when you do so. Many of the notifications are broadcast during [SysHandleEvent](#), which means your application event loop might not have progressed to the point where it is possible for you to display a user interface, or you might overflow the stack by displaying a user interface at this stage. See

the “[Notifications](#)” chapter to learn which notifications are broadcast during SysHandleEvent.

**Compatibility** Implemented only if [Notification Feature Set](#) is present.

## SysNotifyUnregister

**Purpose** Cancel notification of the given event.

**Declared In** NotifyMgr.h

**Prototype** Err SysNotifyUnregister(UInt16 cardNo,  
LocalID dbID, UInt32 notifyType, Int8 priority)

**Parameters**

-> cardNo	Number of the storage card on which the application or code resource resides.
-> dbID	Local ID of the application or code resource.
-> notifyType	The notification for which to unregister. See <a href="#">Notifications</a> .
-> priority	The priority value you passed as part of <a href="#">SysNotifyRegister</a> .

**Result** Returns one of the following error codes:

errNone No error.

sysNotifyErrEntryNotFound

The application wasn't registered to receive this notification.

**Comments** Use this function to remove your code from the list of those that receive notifications about a particular event. This function is particularly necessary if you are writing a shared library or a separately loaded code resource that receives notifications. When the resource is unloaded, it must unregister for all of its notifications, or the system will crash when the notification is broadcast.

**Compatibility** Implemented only if [Notification Feature Set](#) is present.

## Application-Defined Functions

### SysNotifyProcPtr

**Purpose** Handle a notification.

**Declared In** NotifyMgr.h

**Prototype** Err (\*SysNotifyProcPtr)  
(SysNotifyParamType \*notifyParamsP)

**Parameters** -> notifyParamsP  
Pointer to a structure that contains the notification event that occurred and any other information about it. See [SysNotifyParamType](#).

**Result** Always return 0.

**Comments** You pass this function as a parameter to [SysNotifyRegister](#) when registering code that does not have a [PilotMain](#) for a notification. See the description of [SysNotifyRegister](#) for advice on writing a notification handler.

---

**IMPORTANT:** Because the callbackP pointer is used to directly call the function, the pointer must remain valid from the time `SysNotifyRegister` is called to the time the notification is broadcast. If the function is in a shared library, you must keep the library open. If the function is in a separately loaded code resource, the resource must remain locked while registered for the notification. When you close a library or unlock a resource, you must first unregister for any notifications. If you don't, the system will crash when the notification is broadcast.

---

# Overlay Manager

---

This chapter describes the overlay manager API as declared in the header file `OverlayMgr.h`. It discusses the following topics:

- [Overlay Manager Data Structures](#)
- [Overlay Manager Constants](#)
- [Overlay Manager Functions](#)

For more information on the overlay manager, see “[Using Overlays to Localize Resources](#)” on page 365 of the *Palm OS Programmer’s Companion*, vol. I.

## Overlay Manager Data Structures

### OmLocaleType

The `OmLocaleType` struct specifies a locale.

```
typedef struct {
    UInt16    language;
    UInt16    country;
} OmLocaleType;
```

#### Field Descriptions

`language` The language spoken in the locale. This value is one of the `LanguageType` constants.

`country` The country or region where the language is spoken. This value is one of the `CountryType` constants.

#### Compatibility

If Palm OS® [4.0 New Feature Set](#) is present, the `LmLocaleType` replaces `OmLocaleType`. For backward compatibility, `OmLocaleType` is mapped to `LmLocaleType`.

## **Overlay Manager**

### *Overlay Manager Constants*

---

## **Overlay Manager Constants**

<b>Constant</b>	<b>Value</b>	<b>Description</b>
omOverlayRscType	'ovly'	Symbolic name of an overlay resource that is contained in both the base database and the overlay database.
omOverlayRscID	1000	Resource ID of the overlay resource in both the base database and the overlay database.
omFtrCreator	'ovly'	Creator value used for the <code>omFtrShowErrorsFlag</code> feature.
omFtrDefaultLocale	1	Feature that specifies the default locale stored in the ROM. The default locale is used in cases where the system is attempting to open a “stripped” database (one that requires an overlay) and an overlay matching the current locale cannot be found. In this case, the system then looks for an overlay matching the default locale. Use <a href="#">FtrGet</a> and <a href="#">FtrSet</a> to retrieve and set this value.
omFtrShowErrorsFlag	0	Feature that controls the number of error messages displayed by the overlay manager. If this feature is set to true, the overlay manager may display several more error messages when validating an overlay against its base database. This feature only takes effect when the error checking level is set to full (common on debug ROMs, not on release ROMs). Use <a href="#">FtrGet</a> and <a href="#">FtrSet</a> to retrieve and set this value.

---

# Overlay Manager Functions

## OmGetCurrentLocale

**Purpose** Return the current locale.

**Declared In** OverlayMgr.h

**Prototype** void OmGetCurrentLocale  
(LmLocaleType \*currentLocale)

**Parameters** <- currentLocale  
Points to an LmLocaleType struct that identifies the current locale.

**Result** Returns nothing.

**Comments** This function returns the current locale. The current locale controls which overlays are used for resource databases. For example, suppose you have one application and two associated overlays installed, one for US English and one for British English. In this case, if the country specified in the locale returned by this function is cUnitedKingdom, the British English overlay is used for the application. If the country returned is cUnitedStates, the US English overlay is used.

**Compatibility** Implemented only if 3.5 New Feature Set is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call OmGlueGetCurrentLocale. For more information, see Chapter 75, “PalmOSGlue Library.”

**See Also** [OmGetSystemLocale](#)

## OmGetIndexedLocale

**Purpose** Return a system locale by index.

**Declared In** OverlayMgr.h

**Prototype** Err OmGetIndexedLocale (UInt16 localeIndex,  
LmLocaleType \*theLocale)

**Parameters** -> localeIndex Zero-based index of the locale to return.  
<- theLocale Points to an LmLocaleType struct that identifies the locale at that index.

**Result** Returns errNone upon success, or omErrInvalidLocaleIndex if the index is out of bounds.

**Comments** OmGetIndexedLocale is used in a loop to discover how many system overlays are installed for system resources.

If the 4.0 New Feature Set is present, use OmGetNextSystemLocale instead of this function.

OmGetIndexedLocale can be slow on ROMs that contain many valid system locales.

**Compatibility** Implemented only if 3.5 New Feature Set is present.

In Palm OS 3.5, this function would not return a system overlay that was located in RAM. The Palm OS 4.0 version of this function does return system overlays located in RAM.

**See Also** OmGetSystemLocale, OmGetNextSystemLocale

## **OmGetNextSystemLocale**

<b>Purpose</b>	Return a system locale.	
<b>Declared In</b>	<code>OverlayMgr.h</code>	
<b>Prototype</b>	<code>Err OmGetNextSystemLocale (Boolean iNewSearch, OmSearchStateType *ioStateInfoP, LmLocaleType *oLocaleP)</code>	
<b>Parameters</b>	<p>-&gt; <code>iNewSearch</code>      true if this function call is starting a new search, or false if this function call is a continuation of a search.</p> <p>&lt;-&gt; <code>ioStateInfoP</code> If <code>iNewSearch</code> is false, this must point to the same data used for the previous invocation.</p> <p>&lt;- <code>oLocaleP</code>      The found locale.</p>	
<b>Result</b>	Returns <code>errNone</code> if no error or <code>omErrNoNextSystemLocale</code> if no matches were found.	
<b>Comments</b>	<p>You can call this function successively to discover how many system overlays are installed for system resources. Each system overlay found determines a separate valid system locale. Any locale returned by this function can be sent to <a href="#">OmSetSystemLocale</a> to change the system locale.</p> <p>To start the search, pass true for <code>iNewSearch</code>. Allocate an <code>OmSearchStateType</code> structure and pass its address as <code>ioStateInfoP</code>. <code>OmGetNextSystemLocale</code> stores private information in <code>ioStateInfoP</code> and uses it if the search is continued.</p> <p>To continue a search where the previous one left off, pass false for <code>iNewSearch</code> and pass the same <code>ioStateInfoP</code> that you used during the previous call to this function.</p> <p>When called successively, this function eventually returns all system overlays that are in ROM or RAM.</p>	

## **Overlay Manager**

### *Overlay Manager Functions*

---

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## **OmGetRoutineAddress**

**Purpose** Return the address of an overlay manager function.

**Declared In** OverlayMgr.h

**Prototype** void \*OmGetRoutineAddress (OmSelector inSelector)

**Parameters** -> inSelector One of the routine selectors defined in OverlayMgr.h.

**Result** Returns the address of the corresponding function. Returns NULL if an invalid routine selector is passed.

**Comments** You typically use this function to determine whether an overlay manager function exists on the device. As future releases of Palm OS add new functions, older devices with earlier versions of the overlay manager will not implement these newer functions. If OmGetRoutineAddress returns NULL, the function is unavailable.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [IntlGetRoutineAddress](#), [SysGetTrapAddress](#)

## OmGetSystemLocale

**Purpose** Return the system locale.

**Declared In** OverlayMgr.h

**Prototype** void OmGetSystemLocale  
(LmLocaleType \*systemLocale)

**Parameters** <- systemLocale Points to an [LmLocaleType](#) struct that identifies the system locale.

**Result** Returns nothing.

**Comments** You typically don't use this function. Instead, use [OmGetCurrentLocale](#), which returns the locale that determines which overlays are used.

The system locale is saved in the storage heap header and persists across soft resets. When the device is reset, the system locale is used to set the current locale.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call OmGlueGetSystemLocale. For more information, see [Chapter 75, "PalmOSGlue Library."](#)

**See Also** [OmGetCurrentLocale](#)

## Overlay Manager

### Overlay Manager Functions

---

## OmLocaleToOverlayDBName

**Purpose** Return the overlay database's name given the base database name and the locale.

**Declared In** OverlayMgr.h

**Prototype** Err OmLocaleToOverlayDBName  
(const Char \*baseDBName,  
 const LmLocaleType \*targetLocale,  
 Char \*overlayDBName)

**Parameters**

-> baseDBName	The name of the base resource database associated with the overlay.
-> targetLocale	The locale to which this overlay applies. See <a href="#">LmLocaleType</a> . Pass NULL to use the current locale.
<- overlayDBName	The overlay database name given the base database name and the target locale. This buffer must be at least dmDBNameLength bytes.

**Result** Returns errNone upon success, or omErrUnknownLocale if the targetLocale parameter is invalid.

**Comments** The appropriate overlay database name is currently:

*baseDBName\_llCC*

where:

*baseDBName* The name of the base database as you passed it in.

*ll* A two-character code identifying the language.

*CC* A two-character code identifying the country.

The base database name is truncated if necessary to allow for this suffix.

For example, the base database “MemoPad” might have an overlay for US English named “MemoPad\_enUS”.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [OmOverlayDBNameToLocale](#)

## **OmOverlayDBNameToLocale**

**Purpose** Return an overlay database’s locale given its name.

**Declared In** OverlayMgr.h

**Prototype** Err OmOverlayDBNameToLocale  
(const Char \*overlayDBName,  
LmLocaleType \*overlayLocale)

**Parameters** -> overlayDBName  
The name of the overlay database.  
<- overlayLocale  
Points to an [LmLocaleType](#) structure identifying the overlay’s locale.

**Result** Returns errNone upon success, omErrBadOverlayDBName if the string overlayDBName is not long enough to have a locale suffix, or omErrUnknownLocale if the locale cannot be determined.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [OmLocaleToOverlayDBName](#)

## Overlay Manager

### Overlay Manager Functions

---

## OmSetSystemLocale

**Purpose** Set the system locale and reset the device.

**Declared In** OverlayMgr.h

**Prototype** Err OmSetSystemLocale  
(const LmLocaleType \*systemLocale)

**Parameters** -> systemLocale An [LmLocaleType](#) structure specifying the locale to switch the system to.

**Result** Returns errNone upon success, or one of the following if an error occurs:

omErrUnknownLocale  
There is no system overlay for systemLocale.

omErrInvalidLocale  
The system overlay for systemLocale has been found but is invalid.

dmErrInvalidParam  
An error occurred while opening the overlay.

dmErrMemError  
A memory error occurred while opening the overlay.

dmErrDatabaseOpen  
The system overlay was already open.

**Comments** This function changes the system locale to the specified locale if it exists. It first determines that the system overlay exists for the requested locale and that it matches the base system database. If so, it updates the system locale information saved in the storage heap header and resets the device. After the device is reset, the current locale is set to the system locale.

A Palm Powered™ device has a default locale hard-coded into the ROM. This locale is used to set the system locale after a hard reset or any time that the storage heap header is invalid. The storage heap header is typically only invalid when the device is turned on for the first time.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

In Palm OS 3.5, this function would not switch to a system overlay that was located in the RAM. The Palm OS 4.0 version of this function does return system overlays located in the RAM.

**See Also** [OmGetSystemLocale](#)

## **Overlay Manager**

### *Overlay Manager Functions*

---

# Password

---

This chapter provides reference material for the password API. The password API is declared in the header file Password.h.

## Password Functions

### PwdExists

**Purpose** Return true if the system password is set.

**Declared In** Password.h

**Prototype** Boolean PwdExists()

**Parameters** None

**Result** Returns true if the system password is set.

### PwdRemove

**Purpose** Remove the encrypted password string and recover data hidden in databases.

**Declared In** Password.h

**Prototype** void PwdRemove (void)

**Parameters** None

**Result** Returns nothing.

## **Password**

### *Password Functions*

---

## **PwdSet**

**Purpose** Use a passed string as the new password. The password is stored in an encrypted form.

**Declared In** Password.h

**Prototype** void PwdSet (Char\* oldPassword, Char\* newPassword)

**Parameters** oldPassword      The old password must be successfully verified or the new password isn't accepted

                  newPassword      Char\* to a string to use as the password. NULL means no password.

**Result** Returns nothing.

## **PwdVerify**

**Purpose** Verify that the string passed matches the system password.

**Declared In** Password.h

**Prototype** Boolean PwdVerify (Char\* string)

**Parameters** string      String to compare to the system password. NULL means no current password.

The allocated length of string must be at least pwdLength characters long.

**Result** Returns true if the string matches the system password.

# Pen Manager

---

This chapter provides reference material for the pen manager. The pen manager API is declared in the header file PenMgr.h.

For more information on the pen manager, see “[The Pen Manager](#)” on page 62 of the *Palm OS Programmer’s Companion*, vol. I.

## Pen Manager Functions

### PenCalibrate

**Purpose** Set the calibration of the pen.

**Declared In** PenMgr.h

**Prototype** Err PenCalibrate (PointType\* digTopLeftP,  
PointType\* digBotRightP, PointType\* scrTopLeftP,  
PointType\* scrBotRightP)

**Parameters** digTopLeftP      Digitizer output from top-left coordinate.  
digBotRightP      Digitizer output from bottom-right coordinate.  
scrTopLeftP      Screen coordinate near top-left corner.  
scrBotRightP      Screen coordinate near bottom-right corner.

**Result** Returns 0 if no error.

**Comments** Called by Preferences application when calibrating pen.

**See Also** [PenResetCalibration](#)

## **Pen Manager**

### *Pen Manager Functions*

---

## **PenResetCalibration**

**Purpose** Reset the calibration in preparation for calibrating the pen again.

**Declared In** PenMgr.h

**Prototype** Err PenResetCalibration (void)

**Parameters** None.

**Result** Always returns 0.

**Comments** Called by Preferences application before capturing points when calibrating the pen.

**See Also** [PenCalibrate](#)

---

**WARNING!** The digitizer is off after calling this routine and must be calibrated again!

---

# Preferences

---

This chapter describes the preferences API as declared in the header file `Preferences.h`. It discusses the following topics:

- [Preferences Data Types](#)
- [Preferences Constants](#)
- [Preferences Functions](#)

For more information on preferences, see the section “[Preferences](#)” on page 320 of the *Palm OS Programmer’s Companion*, vol. I.

## Preferences Data Types

### MeasurementSystemType

The `MeasurementSystemType` enum specifies the preference for units of measure.

```
typedef enum {
    unitsEnglish = 0,
    unitsMetric
} MeasurementSystemType;
```

#### Value Descriptions

<code>unitsEnglish</code>	The English measurement system (feet, inches, and so on).
<code>unitsMetric</code>	The Metric system (meters, centimeters, and so on).

### SecurityAutoLockType

The `SecurityAutoLockType` enum specifies the values for the auto-locking preference. The auto-locking preference specifies when the device will shut down and lock itself.

## Preferences

### Preferences Data Types

---

```
typedef enum {
    never = 0,
    uponPowerOff,
    atPresetTime,
    afterPresetDelay
} SecurityAutoLockType;
```

#### Value Descriptions

never	The auto-lock feature is disabled.
uponPowerOff	The device locks itself each time it is powered off.
atPresetTime	The device locks itself at a certain time every day.
afterPresetDelay	The device locks itself after a certain amount of idle time.

## SoundLevelTypeV20

The SoundLevelTypeV20 enum specifies whether certain sounds are enabled or disabled. This enum is used as the values for several of the system sound preferences.

```
typedef enum {
    slOn = 0,
    slOff = 1
} SoundLevelTypeV20;
```

#### Value Descriptions

slOn	Enabled
slOff	Disabled

#### Compatibility

This enumerated type is obsolete in Palm OS® versions 3.0 and higher. The preferences that use this enum are replaced by preferences that indicate sound volume as well as whether the sound is on or off.

## SystemPreferencesChoice

The SystemPreferencesChoice enum defines values that you can pass to [PrefGetPreference](#) and [PrefSetPreference](#) to retrieve or set a system preference value.

SystemPreferencesChoice defines one constant for each field in the SystemPreferencesType structure, which should be considered a private structure.

[Table 43.1](#) lists and describes the SystemPreferencesChoice constants. For each constant, it shows what type of data is returned by PrefGetPreference for that constant and which version of the system preferences structure contains this preference.

The system preferences structure keeps track of its own version information because it has been updated several times. Each Palm OS release that updates the system preferences structure increments the structure's version number. [Table 43.1](#) on page 830 specifies which version of the system preferences structure introduced that preference. For each preferences version, there is a constant representing that version. See the section "[Preferences Constants](#)" on page 839 for information on which preferences version corresponds with which Palm OS release. You should check the preference's version number before attempting to obtain the value of any preference introduced after version 2. For example:

```
LmLocaleType currentLocale;
if (PrefGetPreference(prefVersion) >=
    preferenceDataVer9) {
    currentLocale = (LmLocaleType)
    PrefGetPreference(prefLocale);
}
```

Most of the system preferences can be set in the Preferences and Security applications. [Table 43.2](#) on page 838 specifies which system preference is set by each user interface field in these two applications.

## Preferences

### Preferences Data Types

---

**Table 43.1 SystemPreferencesChoice Constants**

Constant	Type	Version	Description
prefVersion	UInt16	2	The preferences version number.
prefCountry	<a href="#">CountryType</a>	2	The country for which the device was built.
prefDateFormat	<a href="#">DateFormatType</a>	2	The short format used to display dates. For example: 95/12/31
prefLongDateFormat	<a href="#">DateFormatType</a>	2	The long format used to display dates. For example: 31 Dec 1995
prefWeekStartDay	Int8	2	The first day of the week (Sunday or Monday). Days of the week are numbered from 0 to 6 starting with Sunday = 0.
prefTimeFormat	<a href="#">TimeFormatType</a>	2	The format used to display time values.
prefNumberFormat	<a href="#">NumberFormatType</a>	2	The format used for numbers, with regards to the thousands separator and the decimal point.
prefAutoOffDuration	UInt8	2	Minutes of user idle time before the device powers off. The default value for this preference is specified by the <code>defaultAutoOffDuration</code> constant.

**Table 43.1 SystemPreferencesChoice Constants (*continued*)**

<b>Constant</b>	<b>Type</b>	<b>Vers ion</b>	<b>Description</b>
			prefAutoOffDuration is replaced by prefAutoOffDurationSecs in version 8 of the preferences structure.
prefSysSoundLevelV20	<a href="#">SoundLevelType</a> <a href="#">V20</a>	2	Specifies whether system sounds are enabled or disabled.
prefGameSoundLevelV20	<a href="#">SoundLevelType</a> <a href="#">V20</a>	2	Specifies whether game sound effects are on or off.
prefAlarmSoundLevelV20	<a href="#">SoundLevelType</a> <a href="#">V20</a>	2	Specifies whether sound alarms are on or off.
prefHidePrivateRecordsV33	Boolean	2	If true, applications should not display database records that have the secret attribute bit set.
prefDeviceLocked	Boolean	2	If true, the device is locked. When the device is locked, it remains so until the user enters the password.
prefLocalSyncRequiresPassword	Boolean	2	If true, the user must enter a password before a HotSync® operation can be performed.
prefRemoteSyncRequiresPassword	Boolean	2	If true, the user must enter a password on the desktop computer before a HotSync operation can be performed.
prefSysBatteryKind	Sys Battery Kind	2	The type of batteries installed. Use <a href="#">SysBatteryInfo</a> to retrieve the battery type instead of this preference.

## Preferences

### Preferences Data Types

---

**Table 43.1 SystemPreferencesChoice Constants (*continued*)**

Constant	Type	Vers ion	Description
prefMinutes WestOfGMT	UInt32	2	The time zone given as minutes <b>east</b> of Greenwich Mean Time (GMT). For preferences version 9 and higher, use prefTimeZone instead.
prefDaylight Savings	<a href="#">DaylightSaving sTypes</a>	2	The type of daylight savings correction. For preferences version 9 and higher, use prefDaylightSavingAdjus tment instead.
prefRonamatic Char	UInt16	2	The virtual character generated when the user enters the ronamatic stroke. The ronamatic stroke is dragging the pen from the Graffiti® area to the top of the screen.
prefHard1Char AppCreator	UInt32	2	The creator ID of the application to be launched by the left-most hard key (the Date Book button by default).
prefHard2Char AppCreator	UInt32	2	The creator ID of the application to be launched by the second hard key from the left (the Address button by default).
prefHard3Char AppCreator	UInt32	2	The creator ID of the application to be launched by the second hard key from the right (the To Do List button by default).

**Table 43.1 SystemPreferencesChoice Constants (*continued*)**

<b>Constant</b>	<b>Type</b>	<b>Version</b>	<b>Description</b>
prefHard4Char AppCreator	UInt32	2	The creator ID of the application to be launched by the right-most hard key (the Memo Pad button by default).
prefCalcChar AppCreator	UInt32	2	The creator ID of the application to be launched by the Calculator silkscreen button.
prefHardCradle CharAppCreator	UInt32	2	The creator ID of the application to be launched by the hard key on the HotSync cradle.
prefLauncher AppCreator	UInt32	2	The creator ID of the application to be launched by the Applications silkscreen button.
prefHardCradle2 CharAppCreator	UInt32	2	The creator ID of the application to be launched by the HotSync button on the modem.
prefAnimation Level	AnimationLevel Type	2	Reserved for future use.
prefSys SoundVolume	UInt16	3	The sound level for system sounds, such as taps and beeps. This is a value from 0 to sndMaxAmp.
prefGame SoundVolume	UInt16	3	The sound level for game sounds. This is a value from 0 to sndMaxAmp.

## Preferences

### Preferences Data Types

---

**Table 43.1 SystemPreferencesChoice Constants (*continued*)**

Constant	Type	Vers ion	Description
prefAlarm SoundVolume	UInt16	3	The sound level for alarms. This is a value from 0 to sndMaxAmp.
prefBeamReceive	Boolean	3	If true, the device can receive beams from other devices. If false, the device cannot receive beams but can still send them.
			This preference is not currently used. Instead, use the <a href="#">ExgControl</a> function with one of the constants described in “ <a href="#">IR Control Constants</a> .”
prefCalibrate DigitizerAtReset	Boolean	3	If true, the user must recalibrate the digitizer after a soft reset. The default is false.
prefSystem KeyboardID	UInt16	4	The resource ID of the keyboard panel.
prefDefSerial PlugIn	UInt32	4	The creator ID of the default serial plug-in database.
prefStayOn WhenPluggedIn	Boolean	5	If true, the device stays powered on when it is in the cradle.
prefStayLit WhenPluggedIn	Boolean	5	If true and prefStayOnWhenPluggedIn is true, the device stays lit when it is in its cradle.
prefAntenna CharAppCreator	UInt32	6	The creator ID of the application to launch when the antenna is raised (used only for devices with built-in antennas).

**Table 43.1 SystemPreferencesChoice Constants (*continued*)**

<b>Constant</b>	<b>Type</b>	<b>Version</b>	<b>Description</b>
prefMeasurementSystem	<a href="#">MeasurementSys</a> <a href="#">temType</a>	7	The system of measurement to use.
prefShowPrivateRecords	<a href="#">privateRecordViewEnum</a>	8	Specifies whether the private records should be displayed, masked, or completely hidden.
prefAutoOffDurationSecs	UInt16	8	Seconds of user idle time before the device powers off. The default value for this preference is specified by the <code>defaultAutoOffDurationSecs</code> constant.
prefTimeZone	Int16	9	The time zone given as minutes east of Greenwich Mean Time (GMT).
			<b>IMPORTANT:</b> Changing the value of this preference does not update the current time.
prefDaylightSavingAdjustment	Int16	9	The number of minutes to add to the current time for daylight savings time.
			<b>IMPORTANT:</b> Changing the value of this preference does not update the current time.
prefTimeZoneCountry	<a href="#">CountryType</a>	9	The country selected to specify what the time zone is.

## Preferences

### Preferences Data Types

---

**Table 43.1 SystemPreferencesChoice Constants (*continued*)**

Constant	Type	Vers ion	Description
prefAutoLockType	<a href="#">SecurityAutoLockType</a>	9	Specifies when the auto-locking feature should take effect. Possibilities are upon power off, at a preset time, or after a certain number of seconds.
prefAutoLockTime	UInt32	9	The time value for the auto-locking feature if the system should lock itself after a delay or at a preset time. Depending on the value of prefAutoLockType, this value is either an absolute date and time given as the number of seconds since January 1, 1904 or a timeout value given as a number of seconds from the current time.
prefAutoLockTimeFlag	Boolean	9	If true, prefAutoLockTime is given in minutes. If false, the time is given in hours.
prefLanguage	<a href="#">LanguageType</a>	9	The language that the device should use.
prefAttentionFlags	<a href="#">AttnFlagsType</a>	9	The user's preferences for receiving attention signals. The returned value is a bit mask that should be tested (using the & operator) with one of the following values:

**Table 43.1 SystemPreferencesChoice Constants (*continued*)**

Constant	Type	Vers ion	Description
			kAttnFlagsUserWantsLED kAttnFlagsUserWants Sound kAttnFlagsUserWants Vibrate kAttnFlagsUserWants CustomEffect
			Note that you can override the values in prefAttentionFlags when you make Attention Manager function calls. See the section “ <a href="#">Getting the User’s Attention</a> ” on page 283 of the <i>Palm OS Programmer’s Companion</i> , vol. I for more information.
prefDefault AppCreator	UInt32	9	Creator ID of the application that is launched after a reset. If 0, the system default application is launched.
prefLocale	<a href="#">LmLocaleType</a>	9	The device’s current locale, which specifies the country and language.
prefDefFepPlugIn Creator	UInt32	10	Creator ID of the default FEP plug-in.
prefColorThemeID	<a href="#">DmResID</a>	10	Resource ID of the color theme.

### Preferences in the User Interface

[Table 43.2](#) shows the SystemPreferencesChoice constants and how they correspond to the values that users can set in the Preferences and Security applications. For further information about each preference, see [Table 43.1](#).

## Preferences

### Preferences Data Types

---

**Table 43.2 Preferences set in standard apps**

<b>Application/Panel</b>	<b>Field</b>	<b>System Preferences Choice Constant</b>
Preferences application General panel	Auto-off After	prefAutoOffDuration, prefAutoOffDurationSecs (Palm OS 3.5 and higher)
	Stay on in Cradle	prefStayOnWhenPluggedIn
	System Sound	prefSysSoundLevelV20, prefSysSoundVolume
	Alarm Sound	prefAlarmSoundLevelV20, prefAlarmSoundVolume
	Alarm Vibrate <sup>1</sup>	prefAttentionFlags
	Alarm LED <sup>a</sup>	prefAttentionFlags
	Game Sound	prefGameSoundLevelV20, prefGameSoundVolume
	Beam Receive field	prefBeamReceive
Preferences application Date & Time panel	Set Time Zone field	prefTimeZone
	Daylight Saving	prefDaylightSavingAdjustment
Preferences application Formats panel	Preset to	prefCountry
	Time	prefDateFormat
	Date	prefDateFormat, prefLongDateFormat
	Week starts	prefWeekStartDay
	Numbers	prefNumberFormat

**Table 43.2 Preferences set in standard apps (*continued*)**

<b>Application/Panel</b>	<b>Field</b>	<b>SystemPreferencesChoice Constant</b>
Preferences application Buttons panel	Buttons on main panel	prefHard1CharAppCreator, prefHard2CharAppCreator, prefHard3CharAppCreator, prefHard4CharAppCreator, prefCalcCharAppCreator, prefLauncherAppCreator
	Pen button	prefRonromaticChar
	HotSync button	prefHardCradleCharApp Creator prefHardCradle2CharApp Creator
Security application	Current Privacy	prefHidePrivateRecordsV33, prefShowPrivateRecords
Security application	Lock & Turn Off button	prefDeviceLocked, prefAutoLockType, prefAutoLockTime, prefAutoLockTimeFlag

1. The Alarm Vibrate and Alarm LED preferences only appear on handhelds running Palm OS 4.0 and later that have the appropriate hardware capabilities. If you install Palm OS 4.0 on an older device, these preferences do not display.

## Preferences Constants

The following table describes the constants defined in the `Preferences.h` header file.

## Preferences

### *Preferences Constants*

---

<b>Constant</b>	<b>Value</b>	<b>Description</b>
defaultAutoLockTime	0	Initial setting for the prefAutoLockTime preference.
defaultAutoLockTimeFlag	0	Initial setting for the prefAutoLockTimeFlag preference.
defaultAutoLockType	never	Initial setting for the prefAutoLockType preference.
defaultAutoOffDuration	2	Initial setting for the prefAutoOffDuration preference, given in minutes.
defaultAutoOffDurationSecs	120	Initial setting for the prefAutoOffDurationSecs preference, given in seconds. The value is 2 times the number of seconds in a minute, or 120 seconds.
defaultAlarmSoundLevel	slOn	Initial setting for the prefAlarmSoundLevelV20 preference.
defaultAlarmSoundVolume	sndMaxAmp	Initial setting for the prefAlarmSoundVolume preference.
defaultGameSoundLevel	slOn	Initial setting for the prefGameSoundLevelV20 preference.
defaultGameSoundVolume	sndMaxAmp	Initial setting for the prefGameSoundVolume preference.
defaultSysSoundLevel	slOn	Initial setting for the prefSysSoundLevelV20 preference.
defaultSysSoundVolume	sndMaxAmp	Initial setting for the prefSysSoundVolume preference.

Constant	Value	Description
noPreferenceFound	-1	The value that <a href="#">PrefGetAppPreferences</a> returns to indicate that preferences couldn't be found.
preferenceDataVer2	2	Version 2 of the system preferences structure, used for Palm OS 2.0.
preferenceDataVer3	3	Version 3 of the system preferences structure, used for Palm OS 3.0.
preferenceDataVer4	4	Version 4 of the system preferences structure, used for Palm OS 3.1.
preferenceDataVer5	5	Version 5 of the system preferences structure, used for Palm OS 3.2.
preferenceDataVer6	6	Version 6 of the system preferences structure, used for Palm OS 3.3.
preferenceDataVer8	8	Version 8 of the system preferences structure, used for Palm OS 3.5.
preferenceDataVer9	9	Version 9 of the system preferences structure, used for Palm OS 4.0.
preferenceDataVer10	10	Version 10 of the system preferences structure, used for Palm OS 5.0.
preferenceDataVer11	11	Version 11 of the system preferences structure, used for Palm OS 5.1.
preferenceDataVerLatest	preferenceDataVer11	The latest version of the system preferences structure.

## Preferences

### Preferences Functions

---

# Preferences Functions

## PrefGetAppPreferences

**Purpose** Return a copy of an application's preferences resource.

**Declared In** Preferences.h

**Prototype** Int16 PrefGetAppPreferences (UInt32 creator,  
UInt16 id, void \*prefs, UInt16 \*prefsSize,  
Boolean saved)

<b>Parameters</b>	-> creator	Creator ID of the application that owns the preferences.
	-> id	ID number of the preferences resource to retrieve. The IDs 0x8000 through 0xFFFF are reserved for system use.
	<- prefs	Pointer to a buffer to hold the preferences.
	<-> prefsSize	Pointer to the size of the prefs buffer passed in. Upon return, contains the number of bytes actually written or the number of bytes needed for the prefs structure.
		To determine the required size for the prefs structure, set prefsSize to 0 and pass NULL for prefs. Upon return, the prefsSize parameter contains the required size. Never set prefs to NULL without also setting prefsSize to 0.
		Always compare the value returned in this parameter with the value you passed in. If the two values differ, you need to resize the prefs structure and call this function again.

---

-> saved	If true, retrieve the preferences from the “saved” preferences database, which is backed up during a HotSync operation. If false, retrieve the preferences from the “unsaved” preferences database, which is usually not backed up during a HotSync operation.
<b>Result</b>	Returns the version number of the retrieved preferences resource, or returns the constant noPreferenceFound if the preferences resource wasn’t found. The returned version number is the same version number that was passed to the <a href="#">PrefSetAppPreferences</a> function.
<b>Comments</b>	<p>Use this function to retrieve the preferences that you previously set with the <a href="#">PrefSetAppPreferences</a> function. You typically call this function in your StartApplication function upon a normal launch. The values of the id and saved parameters should be the same as you specified when calling <a href="#">PrefSetAppPreferences</a>, and the prefs parameter should be a structure of the same type as you passed to <a href="#">PrefSetAppPreferences</a>. Most applications store all preferences in a single preferences resource retrieved by a single call to PrefGetAppPreferences, but this is not required. You can use multiple preferences resources if you wish.</p> <p>Applications typically call PrefGetAppPreferences twice: once with prefs set to NULL and prefsSize set to 0 to determine how much memory to allocate for the preferences structure, and a second time after allocating the required amount of memory to obtain a copy of the application’s preferences resource.</p> <p>The version number returned by this function allows you to handle the case where a new version of the application is being run for the first time. You can compare the value returned by this function with the current version number to determine if you need to set default values for any preferences created by the current release. For more information, see the section “<a href="#">Updating Preferences Upon a New Release</a>” on page 329 of the <i>Palm OS Programmer’s Companion</i>, vol. I.</p>
<b>Compatibility</b>	Implemented only if <a href="#">2.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">PrefSetPreferences</a> , <a href="#">PrefGetAppPreferencesV10</a>

## Preferences

### Preferences Functions

---

## PrefGetAppPreferencesV10

<b>Purpose</b>	Return a copy of an application's preferences.	
<b>Declared In</b>	Preferences.h	
<b>Prototype</b>	Boolean PrefGetAppPreferencesV10 (UInt32 type, Int16 version, void *prefs, UInt16 prefsSize)	
<b>Parameters</b>	-> type	Creator ID of the application that owns the preferences.
	-> version	Version number of the application's preferences.
	<- prefs	Pointer to a buffer to hold preferences.
	-> prefsSize	Size of the buffer passed.
<b>Result</b>	Returns <code>false</code> if the preference resource was not found or the preference resource contains the wrong version number.	
<b>Comments</b>	The content and format of an application preference is application-dependent.	
<b>Compatibility</b>	This function corresponds to the 1.0 version of <code>PrefGetAppPreferences</code> .	
<b>See Also</b>	<a href="#">PrefSetPreferences</a> , <a href="#">PrefGetAppPreferences</a>	

## PrefGetPreference

<b>Purpose</b>	Return a system preference. Use this function instead of <a href="#">PrefGetPreferences</a> .	
<b>Declared In</b>	Preferences.h	
<b>Prototype</b>	<code>UInt32 PrefGetPreference (SystemPreferencesChoice choice)</code>	
<b>Parameters</b>	<code>-&gt; choice</code>	A constant that specifies what preference to retrieve. See <a href="#">SystemPreferencesChoice</a> .
<b>Result</b>	Returns the system preference or 0 if the preference could not be found. On debug ROMs, a non-fatal error message is also displayed if the specified preference cannot be found.	
<b>Comments</b>	This function replaces the 1.0 function <a href="#">PrefGetPreferences</a> . While <a href="#">PrefGetPreferences</a> only lets you retrieve the whole system preferences structure, this function lets you specify which preference to retrieve.	
<b>Compatibility</b>	Implemented only if <a href="#">2.0 New Feature Set</a> is present.	
<b>See Also</b>	<a href="#">PrefSetPreference</a> , <a href="#">PrefGetAppPreferences</a> , <a href="#">PrefGetAppPreferencesV10</a>	

## PrefGetPreferences

<b>Purpose</b>	Return a copy of the system preferences.	
<b>Declared In</b>	Preferences.h	
<b>Prototype</b>	<code>void PrefGetPreferences (SystemPreferencesPtr p)</code>	
<b>Parameters</b>	<code>&lt;- p</code>	Pointer to system preferences.
<b>Result</b>	Returns nothing. Stores the system preferences in p.	

## Preferences

### Preferences Functions

---

- Comments** The `p` parameter points to a memory block allocated by the caller that is filled in by this function.  
This function is often called in `StartApplication` to get localized settings.

---

**NOTE:** This function can only be used to retrieve preferences that were in the 1.0 version of the preferences structure.

---

- See Also** [PrefSetPreferences](#)

## PrefOpenPreferenceDB

- Purpose** Return a handle to either the saved or unsaved preference database.

- Declared In** Preferences.h

- Prototype** DmOpenRef PrefOpenPreferenceDB (Boolean saved)

- Parameters** -> saved If true, open the “saved” preferences database, which is backed up during a HotSync operation. If false, open the “unsaved” preferences database, which usually is not backed up during a HotSync operation.

- Result** Returns the handle, or 0 if an error results.

- Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

- See Also** [PrefGetPreference](#), [PrefSetPreference](#), [PrefOpenPreferenceDBV10](#)

## PrefOpenPreferenceDBV10

<b>Purpose</b>	Return a handle to the system preference database.
<b>Declared In</b>	Preferences.h
<b>Prototype</b>	DmOpenRef PrefOpenPreferenceDBV10 (void)
<b>Parameters</b>	None.
<b>Result</b>	Returns the handle, or 0 if an error results.
<b>Compatibility</b>	This function corresponds to the 1.0 version of PrefOpenPreferenceDB.
<b>See Also</b>	<a href="#">PrefGetPreferences</a> , <a href="#">PrefSetPreferences</a>

## PrefSetAppPreferences

<b>Purpose</b>	Set an application's preferences in the specified preferences database.	
<b>Declared In</b>	Preferences.h	
<b>Prototype</b>	void PrefSetAppPreferences (UInt32 creator, UInt16 id, Int16 version, const void *prefs, UInt16 prefsSize, Boolean saved)	
<b>Parameters</b>	-> creator	Creator ID of the application that owns this preference.
	-> id	ID number of the preference to set. An application can have multiple preferences. The IDs 0x8000 through 0xFFFF are reserved for system use.
	-> version	Version number of the application's preferences.

## Preferences

### Preferences Functions

---

-> prefs	Pointer to a buffer that holds the current value of the preferences structure. Pass NULL if you want to delete the preferences.
-> prefssize	Size of the buffer passed. Pass 0 if you want to delete the preferences structure.
-> saved	If true, saves the preferences in the “saved” preferences database. If not, saves the preferences in the “unsaved” preferences database.

**Result** Returns nothing.

**Comments** You typically call this function in your `StopApplication` function to save the current state of the application or if the user has changed an application preference during the current session.

The “saved” preferences database is backed up when a user performs the HotSync operation. The “unsaved” preferences database is not backed up by default. (The user can use a third-party tool to set the backup bit in the “unsaved” preferences database, which would cause it to be backed up.) Both the “saved” and the “unsaved” preferences reside in the storage heap and thus persist across soft resets. The only way that preferences are lost is if a hard reset is performed. “[Which Preferences Database to Use](#)” on page 327 of the *Palm OS Programmer’s Companion*, vol. I describes how to choose between the “saved” and “unsaved” preferences databases.

The version number that you pass as the `version` parameter is returned when the preferences are retrieved by [`PrefGetAppPreferences`](#). You can use this version number to determine if a new release of the application is being run for the first time. For more information, see the section “[Updating Preferences Upon a New Release](#)” on page 329 of the *Palm OS Programmer’s Companion*, vol. I.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

**See Also** [`PrefSetAppPreferencesV10`](#)

## PrefSetAppPreferencesV10

**Purpose** Save an application's preferences in the preferences database.

**Declared In** Preferences.h

**Prototype** void PrefSetAppPreferencesV10 (UInt32 creator,  
Int16 version, void \*prefs, UInt16 prefsSize)

**Parameters**

-> creator	Creator ID of the application that owns this preference.
-> version	Version number of the application.
-> prefs	Pointer to a buffer holding preferences.
-> prefsSize	Size of the buffer passed.

**Result** Returns nothing.

**Comments** The content and format of an application preference is application-dependent.

**Compatibility** This function corresponds to the 1.0 version of PrefSetAppPreferences.

**See Also** [PrefSetAppPreferences](#), [PrefGetPreferences](#)

## Preferences

### Preferences Functions

---

## PrefSetPreference

**Purpose** Set a system preference. Using this function instead of `PrefSetPreferences` allows you to set selected preferences without having to access the whole structure.

**Declared In** Preferences.h

**Prototype** void PrefSetPreference  
(SystemPreferencesChoice choice, UInt32 value)

**Parameters** -> choice A [SystemPreferencesChoice](#) constant specifying the preference to be set.  
-> value Value to assign to the preference.

**Result** Returns nothing. If the specified preference cannot be found, displays a non-fatal error message on debug ROMs. On release ROMs, this function fails silently.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## PrefSetPreferences

**Purpose** Set the system preferences.

**Declared In** Preferences.h

**Prototype** void PrefSetPreferences (SystemPreferencesPtr p)

**Parameters** -> p Pointer to system preferences.

**Result** Returns nothing.

**Comments** Unless there's a reason for you to access the whole preferences structure, call [PrefSetPreference](#) instead.

---

**NOTE:** This function can only be used to set preferences that were in the 1.0 version of the preferences structure.

---

**See Also** [PrefGetPreferences](#)

## **Preferences**

### *Preferences Functions*

---

# Rectangles

---

This chapter provides reference material for the rectangles API, declared in the header file `Rect.h`. It is divided into the following sections:

- [Rectangle Data Structures](#)
- [Rectangle Functions](#)

## Rectangle Data Structures

### PointType

The `PointType` structure defines a point within a window or on the screen.

```
typedef struct PointType {  
    Coord x;  
    Coord y;  
} PointType;
```

### Field Descriptions

x	Horizontal coordinate.
y	Vertical coordinate.

### RectangleType

The `RectangleType` structure defines a rectangular portion of a window or of the screen.

```
typedef struct RectangleType {  
    PointType topLeft;  
    PointType extent;  
} RectangleType;  
typedef RectangleType *RectanglePtr;
```

## Rectangles

### Rectangle Functions

---

#### Field Descriptions

topLeft	Coordinates of the upper-left corner of the rectangle relative to the window or screen in which the rectangle resides.
extent	Width (extent.x) and height (extent.y) of the rectangle.

## Rectangle Functions

### RctCopyRectangle

**Purpose** Copy the source rectangle to the destination rectangle.

**Declared In** Rect.h

**Prototype** void RctCopyRectangle  
(const RectangleType\* srcRectP,  
 RectangleType\* dstRectP)

**Parameters** srcRectP A pointer to the rectangle to be copied.  
dstRectP A pointer to the destination rectangle.

**See Also** [RctSetRectangle](#)

### RctGetIntersection

**Purpose** Determine the intersection of two rectangles.

**Declared In** Rect.h

**Prototype** void RctGetIntersection  
(const RectangleType\* r1P,  
 const RectangleType\* r2P, RectangleType\* r3P)

**Parameters** r1P A pointer to a source rectangle.

r2P	A pointer to the other source rectangle.
r3P	Upon return, points to a rectangle representing the intersection of r1 and r2.

**Comments** The rectangle type [RectangleType](#), which is pointed to by RectanglePtr, stores the coordinates for the top-left corner of the rectangle plus the rectangle's width and height. This function returns in the r3 parameter a pointer to the rectangle that represents the intersection of the first two rectangles.

If the rectangles r1 and r2 do not intersect, r3 contains a rectangle whose top-left coordinate is the maximum of r1 and r2's top-left coordinates and whose extent varies based on the location of the two rectangles.

**Compatibility** On releases prior to Palm OS® 3.5, if rectangles r1 and r2 don't intersect, r3 contains a rectangle that begins at coordinates (0,0) and has 0 width and 0 height. On Palm OS 3.5 and later, if the two rectangles don't intersect then r3 contains a rectangle in which one or both of the extent coordinates is zero.

## RctInsetRectangle

**Purpose** Move all of the boundaries of a rectangle by a specified offset.

**Declared In** Rect.h

**Prototype** void RctInsetRectangle (RectangleType\* rP,  
Coord insetAmt)

**Parameters** rP A pointer to the rectangle.  
insetAmt Number of pixels to move the boundaries. This can be a negative number.

**Comments** The rectangle type [RectangleType](#), which is pointed to by RectanglePtr, stores the coordinates for the top-left corner of the rectangle plus the rectangle's width and height. This function adds insetAmt to the x and y values of the top-left coordinate and then

## Rectangles

### Rectangle Functions

---

adjusts the width and the height accordingly so that all of the sides of the rectangle are contracted or expanded by the same amount.

A positive `insetAmt` creates a smaller rectangle that is contained inside the old rectangle's boundaries. A negative `insetAmt` creates a larger rectangle that surrounds the old rectangle.

**See Also** [RctOffsetRectangle](#)

## RctOffsetRectangle

**Purpose** Move the top and left boundaries of a rectangle by the specified values.

**Declared In** Rect.h

**Prototype** void RctOffsetRectangle (RectangleType\* rP,  
Coord deltaX, Coord deltaY)

**Parameters**

rP	A pointer to the rectangle.
deltaX	Number of pixels to move the left boundary. This can be a negative number.
deltaY	Number of pixels to move the top boundary. This can be a negative number.

**Comments** The rectangle type [RectangleType](#), which is pointed to by `RectanglePtr`, stores the coordinates for the top-left corner of the rectangle plus the rectangle's width and height. This function adds `deltaX` to the x value of the top-left coordinate and `deltaY` to the y value. The width and height are unchanged. Thus, this function shifts the position of the rectangle by the `deltaX` and `deltaY` amounts.

**See Also** [RctInsetRectangle](#)

## RctPtInRectangle

**Purpose** Determine if a point lies within a rectangle's boundaries.

**Declared In** Rect.h

**Prototype** Boolean RctPtInRectangle (Coord x, Coord y,  
const RectangleType\* rP)

**Parameters** x The x coordinate of the point.  
y The y coordinate of the point.  
rP The rectangle.

**Result** Returns true if the point (x, y) lies within the boundaries of rectangle r, false otherwise.

## RctSetRectangle

**Purpose** Sets a rectangle's values.

**Declared In** Rect.h

**Prototype** void RctSetRectangle (RectangleType\* rP,  
Coord left, Coord top, Coord width, Coord height)

**Parameters** rP A pointer to the rectangle to be set.  
left The x value for the top-left coordinate of the rectangle.  
top The y value for the top-left coordinate of the rectangle.  
width The rectangle's width.  
height The rectangle's height.

**See Also** [RctCopyRectangle](#)

## **Rectangles**

### *Rectangle Functions*

---

# Sound Manager

---

This chapter describes the API that's defined by the Sound Manager. You use this API to make and record sounds.

The Sound Manager controls two independent sound facilities:

- Single voice, monophonic, square-wave sound synthesis, useful for system beeps. This is the traditional (pre-OS 5) PalmSource sound; let's call it "simple sound."
- Stereo, multi-format, sampled data recording and playback (new in Palm OS 5). We'll call it "sampled sound."

The API for these two facilities are entirely separate; the simple sound API is described first...

- [Simple Sound Structures and Constants](#)
- [Simple Sound Functions](#)
- [Simple Sound Application-Defined Functions](#)

...followed by the sampled sound API.

- [Sampled Sound Structures, Constants, and Data Types](#)
- [Sampled Sound Functions](#)
- [Sampled Sound Application-Defined Functions](#)

All elements described here are declared in `SoundMgr.h`.

## Overview

There are three ways to play a simple sound:

- You can play a single tone of a given pitch, amplitude, and duration by calling [SndDoCmd](#).
- You can play a pre-defined system sound through [SndPlaySystemSound](#).
- You can play a tune by passing in a Level 0 Standard MIDI file (SMF) through the [SndPlaySmf](#) function.

## Sound Manager

### Simple Sound Structures and Constants

---

There are a handful of other simple sound functions, but they mostly support these three fundamental functions.

Over in the sampled sound facilities, there are two fundamental functions:

- [SndStreamCreate](#) opens a new sampled sound “stream” from/into which you record/playback buffers of “raw” data. The trick is that you first have to configure the stream to tell it how to interpret the data.
- [SndPlayResource](#) is used to playback sound data that’s read from a (formatted) sound file. The function configures the playback stream for you, based on the format information in the sound file header. Currently, only uncompressed WAV and IMA ADPCM WAV formats are recognized (this also known as DVI ADPCM). Note that SndPlayResource is *only* used to play back sound; it can’t be used for recording.

The Sound Manager also provides functions that let you set the volume and stereo panning for individual recording and playback streams. See [SndStreamSetVolume](#) and [SndStreamSetPan](#).

For more information on the Sound Manager, see the section “[Sound](#)” in the *Palm OS Programmer’s Companion*, vol. I.

## Simple Sound Structures and Constants

### SndCallbackInfoType

Structure that encapsulates a callback function and its argument data. SndCallbackInfoType is used by the [SndSmfCallbacksType](#) structure, which is used to list the callback functions that are called during SMF playback.

```
typedef struct SndCallbackInfoType {  
    MemPtr funcP;  
    UInt32 dwUserData;  
} SndCallbackInfoType;
```

The fields are:

---

funcP	A pointer to the callback function.
dwUserData	Data that's passed as an argument to the function.

---

## SndCmdIDTag

The SndCmdIDTag enumeration contains constants that represent specific sound operations used in simple sound playback.

```
typedef enum SndCmdIDTag {
    sndCmdFreqDurationAmp = 1,
    sndCmdNoteOn,
    sndCmdFrqOn,
    sndCmdQuiet
};
```

See [SndDoCmd](#) for descriptions of the operations that are represented by these values.

## SndCommandType

The SndCommandType structure encapsulates a sound synthesis operation and its associated parameters. Used by the [SndDoCmd](#) function.

```
typedef struct SndCommandType {
    SndCmdIDType cmd;
    UInt8 reserved;
    Int32 param1;
    UInt16 param2;
    UInt16 param3;
} SndCommandType;
```

## Sound Manager

### Simple Sound Structures and Constants

---

The fields are:

cmd	Constant that represents a sound operation. The operations are listed and described in <a href="#">SndDoCmd</a> .
reserved	Don't touch this one.
param1, param2, param3	Operation-specific parameters. The parameters' meanings are described in <a href="#">SndDoCmd</a> .

### **sndMaxAmp**

Constant that defines the upper limit for simple sound amplitude values. Use this value with the simple sound API only (such as [SndDoCmd](#)). The sndMaxAmp value is *not* compatible with the sampled sound amplitude range.

```
#define sndMaxAmp 64
```

### **SndMidiListItemType**

Structure that locates a MIDI file. The structure is used by the [SndCreateMidiList](#) function.

```
typedef struct SndMidiListItemType{
    Char   name [sndMidiNameLength];
    UInt32 uniqueRecID;
    MemHandle dbH;
} SndMidiListItemType;
```

The fields are:

name	The null-terminated name of the MIDI file.
uniqueRecID	The ID of the record that holds the MIDI file.
dbH	Handle to the database that holds the record.

## **sndMidiNameLength**

Constant that defines the maximum string length (including the null terminator) for a MIDI file or MIDI track name.

```
#define sndMidiNameLength 32
```

## **SndMidiRecHdrType**

Structure that encapsulates the header of a MIDI record.

```
typedef struct SndMidiRecHdrType {  
    UInt32  signature;  
    UInt8   bDataOffset;  
    UInt8   reserved;  
} SndMidiRecHdrType;
```

The fields are:

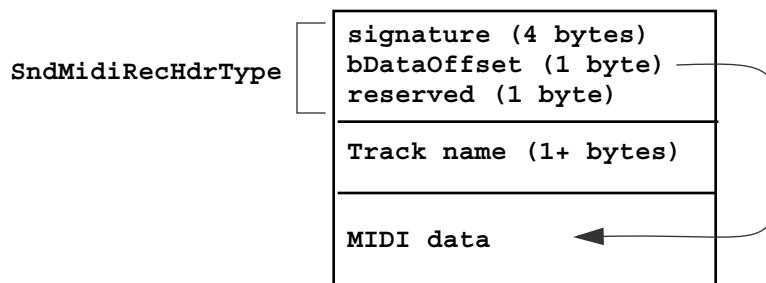
---

signature	Always set to <a href="#">sndMidiRecSignature</a> .
bDataOffset	Offset, in bytes, from the beginning of the record to the first byte of the MIDI data.
reserved	Always set to 0.

---

[Figure 45.1](#) depicts a complete Palm OS® MIDI record.

**Figure 45.1 Palm OS Midi Record**



To get to the track name, you have to bump the structure pointer yourself. For example:

## Sound Manager

### *Simple Sound Structures and Constants*

---

```
Char *trackName = (Char *)myMidiRecHdr + sizeof(SndMidiRecHdrType);
```

---

The MIDI track name is null-terminated, even if it's empty. It's at least one byte long and at most [sndMidiNameLength](#) bytes long.

### **sndMidiRecSignature**

Endian-cognizant constant that's used to tag a MIDI record. The constant is used as the value of the `signature` field of the [SndMidiRecHdrType](#) structure.

```
#if CPU_ENDIAN == CPU_ENDIAN_BIG
#define sndMidiRecSignature 'PMrc'
#else
#define sndMidiRecSignature 'crMP'
```

### **SndSmfCallbacksType**

Structure that contains a set of application-defined functions that are called during MIDI playback. To register your callback functions, call [SndPlaySmf](#).

```
typedef struct SndSmfCallbacksType {
    SndCallbackInfoType completion;
    SndCallbackInfoType blocking;
    SndCallbackInfoType reserved;
} SndSmfCallbacksType;
```

The fields are:

---

completion	Completion function; see <a href="#">SndComplFuncType</a> .
blocking	Blocking function; see <a href="#">SndBlockingFuncType</a> .
reserved	Reserved. Set this field to 0.

---

## SndSmfChanRangeType

Structure that defines the range of enabled MIDI channels. Events on MIDI channels outside the enabled range are ignored. By default, no channels are enabled.

```
typedef struct SndSmfChanRangeType {  
    UInt8 bFirstChan;  
    UInt8 bLastChan;  
} SndSmfChanRangeType;
```

The fields are:

---

bFirstChan	The first enabled channel in the range [0, 15].
bLastChan	The last enabled channel in the range [0, 15].

---

---

**WARNING!** The SndSmfChanRangeType structure expects MIDI channels to be in the range [0, 15]; real MIDI channel values are in the range [1, 16]. Thus, PalmSource MIDI channel 0 is real MIDI channel 1, PalmSource MIDI channel 1 is real MIDI channel 2, and so on.

---

## SndSmfCmdEnum

Constants used to tell [SndPlaySmf](#) whether to set or retrieve data.

```
typedef enum SndSmfCmdEnumTag {  
    sndSmfCmdPlay = 1,  
    sndSmfCmdDuration,  
} SndSmfCmdEnum;
```

See [SndPlaySmf](#) for descriptions of these values.

## SndSmfOptionsType

Structure that establishes MIDI performance parameters. This structure is used in the [SndPlaySmf](#) function to establish new parameter settings or to return the currently set values, depending

## Sound Manager

### *Simple Sound Structures and Constants*

---

on how the function is called. In the case where the structure returns values, only the “performance marker” fields (dwStartMilliSec and dwEndMilliSec) are valid.

```
typedef struct SndSmfOptionsType {
    UInt32 dwStartMilliSec;
    UInt32 dwEndMilliSec;
    UInt16 amplitude;
    Boolean interruptible;
    UInt8 reserved1;
    UInt32 reserved;
} SndSmfOptionsType;
```

The fields are

---

dwStartMilliSec	The “beginning of performance” marker, measured in milliseconds from the beginning of the track. A value of 0 plays the track from the beginning. The time difference between dwStartMilliSec and the performance time of the first subsequent MIDI event is respected. For example, if dwStartMilliSec is 2000 and the first (subsequent) note-on event is at 3000, there will be a 1000 millisecond “pause” before the note is played.
dwEndMilliSec	The “end of performance” marker, measured in milliseconds from the beginning of the track. To play to the end of the track, set this to <a href="#">sndSmfPlayAllMilliSec</a> .

---

amplitude	Specifies the volume of the track, in the range [0, sndMaxAmp]. The default is sndMaxAmp. If set to 0, the MIDI file isn't played.
interruptible	If true (the default), MIDI playback is interrupted if the user interacts with the controls (digitizer, buttons, etc.), even if the interaction doesn't generate a sound command. If false, playback is not interrupted.
reserved1	Reserved.
reserved	Reserved. Set this field to 0.

---

## **sndSmfPlayAllMilliSec**

Represents the (temporal) far end of a MIDI file. You can use this constant as the value of the dwEndMilliSec field of the [SndSmfOptionsType](#) structure before passing the structure to [SndPlaySmf](#). This setting tells the function to play the entire file.

```
#define sndSmfPlayAllMilliSec 0xFFFFFFFFFUL
```

## **SndSysBeepTag**

Constants that represent a set of pre-defined system beeps. See [SndPlaySystemSound](#) for more information on the system beeps and their intended uses.

```
typedef enum SndSysBeepTag {  
    sndInfo = 1,  
    sndWarning,  
    sndError,  
    sndStartUp,  
    sndAlarm,  
    sndConfirmation,  
    sndClick  
} ;
```

# Simple Sound Functions

## SndCreateMidiList

**Purpose** Generates a list of MIDI records.

**Declared In** SoundMgr.h

**Prototype** Boolean SndCreateMidiList (UInt32 creator,  
Boolean multipleDBs, UInt16\* recordCount,  
MemHandle \*recordList)

**Parameters**

-> creator	Creator ID of the database in which the function looks for MIDI records. Pass 0 to search all databases.
-> multipleDBs	Pass true to search multiple databases for MIDI records. Pass false to search only in the first database that meets the search criteria.
<- recordCount	Returns the number of MIDI records that were found.
<- recordList	Returns a pointer to an array of <a href="#">SndMidiListItemType</a> structures, one structure for each record that was found.

**Result** Return true if records were found, otherwise returns false.

**Compatibility** Implemented in Palm OS 3.0 and later.

## SndDoCmd

**Purpose** Asks the Sound Manager to perform a simple sound synthesis operation.

**Prototype** Err SndDoCmd (void\* channel,  
SndCommandPtr command, Boolean noWait)

**Parameters** -> channel A pointer to the sound channel on which you want to perform the operation. Pass NULL for the “shared” sound channel.

---

**IMPORTANT:** The Sound Manager only supports one channel of sound synthesis: You must pass NULL as the value of channel.

---

-> command Pointer to a [SndCommandType](#) that describes the operation and associated parameters. See **Comments**, below, for information on the sound commands.

-> noWait Sets the function to be asynchronous (true) or synchronous (false) with respect to the caller.

---

**IMPORTANT:** SndDoCmd is always synchronous: The noWait value is currently ignored.

---

**Result** errNone Success.

  sndErrBadParam  
                    Invalid parameter.

  sndErrBadChannel  
                    Invalid channel pointer.

  sndErrQFull  
                    The sound queue is full.

## Sound Manager

### *Simple Sound Functions*

---

**Comments** The sound operations that are performed by SndDoCmd are encapsulated in the [SndCommandType](#) structure:

```
typedef struct SndCommandType {  
    SndCmdIDType cmd;  
    UInt8 reserved;  
    Int32 param1;  
    UInt16 param2;  
    UInt16 param3;  
} SndCommandType;
```

The cmd field represents the operation; the paramN fields are data that's passed to the operation. The operations and data that SndDoCmd supports are described below:

cmd Value	param Values
sndCmdFreqDurationAmp plays a tone. The function blocks until the tone has finished.	param1 is the tone's frequency in Hertz. param2 is its duration in milliseconds param3 is its amplitude in the range [0, sndMaxAmp]. If the amplitude is 0, the sound isn't played and the function returns immediately.
sndCmdFrqOn initiates a tone. The function returns immediately while the tone plays in the background. Subsequent sound playback requests will interrupt the tone.	Same as the parameters for sndCmdFreqDurationAmp.

<b>cmd Value</b>	<b>param Values</b>
sndCmdNoteOn initiates a MIDI-defined tone. The function returns immediately while the tone plays in the background. Subsequent sound playback requests will interrupt the tone.	param1 is the tone's pitch given as a MIDI key number in the range [0, 127]. param2 is the tone's duration in milliseconds param3 is its amplitude given as MIDI velocity [0, 127].
sndCmdQuiet stops the playback of the currently generated tone.	Ignored.

**Compatibility** Commands other than `sndCmdFreqDurationAmp` are implemented in Palm OS 3.0 and later. In OS versions earlier than 3.0, `SndDoCmd` will crash with a fatal error if you pass a command other than `sndCmdFreqDurationAmp`.

**See Also** [SndPlaySmf](#)

## SndGetDefaultVolume

**Purpose** Returns volume levels cached by the Sound Manager.

**Prototype** `void SndGetDefaultVolume (UInt16 *alarmAmp,  
                  UInt16 *sysAmp, UInt16 *masterAmp)`

<b>Parameters</b>	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 33%; padding-right: 10px;">&lt;- <code>alarmAmp</code></td> <td style="width: 33%; padding-right: 10px;">Pointer to the alarm amplitude.</td> </tr> <tr> <td>&lt;- <code>sysAmp</code></td> <td>Pointer to the system sound amplitude.</td> </tr> <tr> <td>&lt;- <code>masterAmp</code></td> <td>Pointer to the master amplitude.</td> </tr> </table>	<- <code>alarmAmp</code>	Pointer to the alarm amplitude.	<- <code>sysAmp</code>	Pointer to the system sound amplitude.	<- <code>masterAmp</code>	Pointer to the master amplitude.
<- <code>alarmAmp</code>	Pointer to the alarm amplitude.						
<- <code>sysAmp</code>	Pointer to the system sound amplitude.						
<- <code>masterAmp</code>	Pointer to the master amplitude.						

**Comments** Pass NULL for settings that you don't care about.

**Compatibility** Never call this function in Palm OS 5. To retrieve default volume levels, you should ask for the user's preferences settings.

**See Also** [SndSetDefaultVolume](#)

## Sound Manager

### *Simple Sound Functions*

---

## SndInterruptSmfIrregardless

<b>Purpose</b>	Stops the currently playing MIDI file, even if the performance was declared to be intolerant of interruptions.
<b>Prototype</b>	Err SndInterruptSmfIrregardless (void)
<b>Result</b>	Always returns errNone (success).
<b>Compatibility</b>	Implemented in Palm OS 4.0 and later. The name is incompatible with the English language.
<b>See Also</b>	<a href="#">SndPlaySmf</a> , <a href="#">SndPlaySmfResourceIrregardless</a>

## SndPlaySmf

<b>Purpose</b>	Performs a Standard MIDI File, or returns the duration of the file.
<b>Prototype</b>	Err SndPlaySmf (void *channel, SndSmfCmdEnum command, UInt8 *midiData, SndSmfOptionsType *options, SndSmfChanRangeType *channelRange, SndSmfCallbacksType *callbacks, Boolean noWait)
<b>Parameters</b>	-> channel A pointer to the sound channel on which you want to perform the MIDI file. Pass NULL for the “shared” sound channel.

---

**IMPORTANT:** The Sound Manager only supports one channel of sound synthesis: You must pass NULL as the value of channel.

---

-> command	Either SndSmfCmdPlay (play the file) or SndSmfCmdDuration (return the duration of the file in milliseconds).
-> midiData	The MIDI data; this can point to an <a href="#">SndMidiRecHdrType</a> struct, or it can point

		directly to the actual MIDI data bytes in memory.
-> options		A pointer to a <a href="#">SndSmfOptionsType</a> that defines performance parameters (volume, starting offset, interruption tolerance). For default behavior, pass NULL. For more information (including default settings), see <a href="#">SndSmfOptionsType</a> .
-> channelRange		A pointer to a <a href="#">SndSmfChanRangeType</a> that specifies the range of MIDI channels (in the SMF data) to use during playback. To play all channels, pass NULL.
-> callbacks		A pointer to a <a href="#">SndSmfCallbacksType</a> that holds your callback functions. Pass NULL if you don't want any callbacks.
noWait		This value is ignored. This function always finishes playing the SMF selection before returning (but see <b>Comments</b> , below).
<b>Result</b>		
errNone		Success.
sndErrBadParam		Invalid value passed to this function.
sndErrBadChannel		Invalid sound channel.
sndErrMemory		Insufficient memory.
sndErrOpen		Tried to open channel that's already open.
sndErrQFull		Can't accept more notes.
sndErrFormat		Unsupported data format.
sndErrBadStream		Invalid data stream.
sndErrInterrupted		Play was interrupted.

## Sound Manager

### Simple Sound Functions

---

<b>Comments</b>	Although this call is always synchronous, you can register a “blocking” function that’s called periodically as the MIDI file is playing. See <a href="#">SndBlockingFuncType</a> for more information.  Normally, playback is halted by events generated by user interaction with the screen, digitizer, or hardware-based buttons. You can override this behavior by setting the <code>interruptible</code> field of the <code>options</code> argument to <code>false</code> .  This function waits until any currently playing simple sound has finished before starting playback of the requested MIDI data. A similar function, <a href="#">SndPlaySmfIrregardless</a> , doesn’t wait: It interrupts the current performance and immediately begins playback of the requested data.
<b>Compatibility</b>	Implemented in Palm OS 3.0 and later.

## SndPlaySmfIrregardless

<b>Purpose</b>	Like <a href="#">SndPlaySmf</a> , but interrupts any currently playing simple sound, regardless of that sound’s declared interruption tolerance.
<b>Prototype</b>	<pre>Err SndPlaySmfIrregardless (void *channel, SndSmfCmdEnum command, UInt8 *midiData, SndSmfOptionsType *options, SndSmfChanRangeType *channelRange, SndSmfCallbacksType *callbacks, Boolean noWait)</pre>
<b>Comments</b>	For further information, see <a href="#">SndPlaySmf</a> .
<b>Compatibility</b>	Implemented in Palm OS 4.0 and later.

## SndPlaySmfResource

<b>Purpose</b>	Plays a MIDI track read out of an open resource database.	
<b>Prototype</b>	<pre>Err SndPlaySmfResource (UInt32 resType, Int16 resID, SystemPreferencesChoice volume)</pre>	
<b>Parameters</b>	-> resType	SMF resource type.
	-> resID	SMF resource ID.
	-> volume	Volume setting; one of:  prefSysSoundVolume prefGameSoundVolume prefAlarmSoundVolume
<b>Result</b>	errNone	Success.
	sndErrBadParam	The volumeSelector parameter has an invalid value or the SMF resource has invalid data.
	dmErrCantFind	The specified resource doesn't exist.
	<i>other values</i>	See <a href="#">SndPlaySmf</a>
<b>Comments</b>	<p>This function plays the entire MIDI file using all MIDI channels. Playback is interrupted by a key down or digitizer event. No callbacks are specified.</p> <p>This function waits until any currently playing simple sound has finished before starting playback of the requested MIDI data. A similar function, <a href="#">SndPlaySmfResourceIrregardless</a>, doesn't wait: It interrupts the current performance and immediately begins playback of the requested data.</p>	
<b>Compatibility</b>	Implemented in Palm OS 3.2 and later.	

## SndPlaySmfResourceIrregardless

**Purpose** Like [SndPlaySmfResource](#), but interrupts any currently playing simple sound, regardless of that sound's declared tolerance for interruption.

**Prototype** Err SndPlaySmfResourceIrregardless  
(UInt32 resType, Int16 resID,  
SystemPreferencesChoice volumeSelector)

**Comments** For further information, see [SndPlaySmfResource](#).

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## SndPlaySystemSound

**Purpose** Plays a pre-defined (simple) system sound.

**Prototype** void SndPlaySystemSound (SndSysBeepType beepID)

**Parameters** -> beepID      One of the system beep sound constants, listed below.

**Comments** The system beep sounds are represented by the SndSysBeepType constants:

- **sndInfo.** Heralds non-crucial information.
- **sndWarning.** Grabs the user's attention
- **sndError.** Indicates an illegal operation.
- **sndStartUp.** Played at device start up time.
- **sndAlarm.** Generic alarm sound; note that this is *not* the Datebook's alarm sound.
- **sndConfirmation.** Indicates approval or acceptance.
- **sndClick.** The button click sound.

If you're playing an alarm (sndAlarm), the user's alarm volume preference setting is used. For all other system sounds, the system volume preference is used.

In addition, `sndAlarm` sounds are played synchronously (the function blocks). All other sounds are played asynchronously.

## SndSetDefaultVolume

**Purpose** Sets the default sound volume levels cached by the Sound Manager.

**Prototype** `void SndSetDefaultVolume (UInt16 *alarmAmp,  
                  UInt16 *sysAmp, UInt16 *masterAmp)`

**Parameters** `-> alarmAmp` Pointer to the alarm amplitude.  
`-> sysAmp` Pointer to the system sound amplitude.  
`-> masterAmp` Pointer to the master amplitude.

**Result** Returns nothing.

**Comments** Any of the arguments may be `NULL`. In that case, the corresponding setting is not altered.

**Compatibility** Never call this function in Palm OS 5.

**See Also** [SndGetDefaultVolume](#)

# Simple Sound Application-Defined Functions

## SndBlockingFuncType

**Purpose** Invoked periodically during SMF playback.

**Prototype** `Boolean SndBlockingFuncType (void* channel,  
                  UInt32 userData, Int32 time)`

**Parameters** `-> channel` A pointer to the sound channel on which the file is being played. Currently always `NULL`.  
`-> userData` Arbitrary data that's defined when the callback function is registered.

## Sound Manager

### Simple Sound Application-Defined Functions

---

-> time      The amount of time, in milliseconds, available for completion of this function.

**Result**    If the function returns `true`, playback continues. If it returns `false`, playback is aborted.

**Comments**    Your `SndBlockingFuncType` function is called whenever the MIDI parser is “between notes.” You can do whatever you want during this period, as long as it doesn’t take more than `time` milliseconds.

**See Also**    [SndComplFuncType](#) for an example of how to register MIDI callback functions.

## SndComplFuncType

**Purpose**    Invoked immediately after a MIDI file (SMF) finishes playing.

**Prototype**    `void SndComplFuncType (void* channel,  
                  UInt32 userData)`

**Parameters**    -> `channel`      A pointer to the sound channel on which the file was played; currently always `NULL`.  
                  -> `userData`      Caller-defined data that’s copied when the callback is registered.

**Comments**    To register a `SndComplFuncType` function (and the `userData` that’s fed to it), you define a [SndCallbackInfoType](#) structure, load that structure into the `completion` field of a [SndSmfCallbacksType](#) structure, and then pass *that* structure to [SndPlaySmf](#).

The following example shows how to register a completion function as well as an SMF “blocking” function ([SndBlockingFuncType](#)).

---

```
/* First, we create and load two SndCallbackInfoType structs. We assume that
MyCompletionFunc is a valid SndComplFuncType implementation, and that
MyBlockingFunc is a valid SndBlockingFuncType.
*/
```

```
SndCallbackInfoType completionFunc, blockingFunc;  
completionFunc.funcP = MyCompletionFunc;  
completionFunc.dwUserData = 47;  
blockingFunc.funcP = MyBlockingFunc;  
blockingFunc.dwUserData = (UInt32)&(char *) "we're done";  
  
/* Next, we create an SndSmfCallbacksType struct and load the previous structs  
into it.  
*/  
SndSmfCallbacksType callbacks;  
callbacks.completion = completionFunc  
callbacks.blocking = blockingFunc;  
  
/* Finally, we pass the callbacks into SndPlaySmf.  
*/  
SndPlaySmf(NULL, sndSmfCmdPlay, smf, NULL, NULL, callbacks, true);
```

---

## Sampled Sound Structures, Constants, and Data Types



### SndPtr

Type used to cast a pointer to the sound data used by [SndPlayResource](#).

```
typedef void *SndPtr;
```

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.



### SndSampleType

Data type that's used for [SndSampleTypeTag](#) values.

```
typedef Enum16 SndSampleType;
```

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.



## New **SndSampleTypeTag**

Constants that represent the sample format (size, data type, endianness) of a sampled sound stream. Used by [SndStreamCreate](#).

The enumeration is defined as:

```
typedef enum SndSampleTypeTag {
    sndInt8 = 0x01,
    sndUInt8 = 0x11,
    sndInt16Big = 0x02,
    sndInt16Little = 0x12,
    sndInt32Big = 0x04,
    sndInt32Little = 0x14,
    sndFloatBig = 0x24,
    sndFloatLittle = 0x34,
#if CPU_ENDIAN == CPU_ENDIAN_BIG
    sndInt16 = sndInt16Big,
    sndInt16Opposite = sndInt16Little,
    sndInt32 = sndInt32Big,
    sndInt32Opposite = sndInt32Little,
    sndFloat = sndFloatBig,
    sndFloatOpposite= sndFloatLittle
#else
    sndInt16 = sndInt16Little,
    sndInt16Opposite = sndInt16Big,
    sndInt32 = sndInt32Little,
    sndInt32Opposite= sndInt32Big,
    sndFloat = sndFloatLittle,
    sndFloatOpposite= sndFloatBig
#endif
};
```

The constants are:

---

sndInt8, sndUInt8	Signed and unsigned 8-bit data.
sndInt16, sndInt32, sndFloat	16-bit integer, 32-bit integer, and floating point data in the device's native endianness.

sndInt16Opposite, sndInt32Opposite, sndFloatOpposite	The same as the above, but byte-swapped.
sndInt16Big, sndInt32Big, sndFloatBig	The same as the above, but big- endian.
sndInt16Little, sndInt32Little, sndFloatLittle	The same as the above, but little-endian.

---

Note that the lower four bits of these constants gives the size (in bytes) of a single sample, thus:

---

```
UInt8 byteSize = formatConstant & 0x0f
```

---

**NOTE:** In Palm OS 5, the 32-bit and floating point formats aren't supported.

---

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.



## SndStreamMode

Data type that's used for [SndStreamModeTag](#) values.

```
typedef Enum8 SndStreamMode;
```

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.



## SndStreamModeTag

Constants that represent the "direction" (input or output) of a sampled sound stream. Used by the [SndStreamCreate](#) function.

```
typedef enum SndStreamModeTag {
```

## Sound Manager

*Sampled Sound Structures, Constants, and Data Types*

---

```
    sndInput,  
    sndOutput  
};
```

The constants are:

---

sndInput	Input stream used for recording.
sndOutput	Output stream used for playback.

---

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.



### SndStreamRef

Data type that represents a sampled stream. You create an SndStreamRef through [SndStreamCreate](#).

```
typedef UInt32 SndStreamRef;
```

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.



### SndStreamWidth

Data type that's used for [SndStreamWidthTag](#) values.

```
typedef Enum8 SndStreamWidth;
```

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.



### SndStreamWidthTag

Constants that represent mono and stereo sampled data streams. Used by the [SndStreamCreate](#) function.

```
typedef enum SndStreamWidthTag {  
    sndMono,  
    sndStereo  
};
```

The constants are:

---

sndMono	Mono (one channel) stream.
sndStereo	Stereo (two channel) stream.

---

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.



## Sound Resource Playback Flags

Flags used by [SndPlayResource](#).

```
#define sndFlagSync 0x00000000  
#define sndFlagAsync 0x00000001  
#define sndFlagNormal sndFlagSync
```

See [SndPlayResource](#) for information on these flags.

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.



## Stereo Pan Constants

The stereo pan settings can be used in [SndStreamSetPan](#).

```
#define sndPanCenter (0)  
#define sndPanFullLeft (-1024)  
#define sndPanFullRight (1024)
```

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.

## Sound Manager

### *Sampled Sound Functions*

---



New

### Volume Constants

The volume constants can be used by [SndStreamSetVolume](#) and [SndPlayResource](#). The constants tell the functions to retrieve the named sound volume preference (as set by the user) and apply it as a volume setting.

```
enum
{
    sndSystemVolume = -1,
    sndGameVolume = -2,
    sndAlarmVolume = -3
};
```

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.

## Sampled Sound Functions



New

### SndPlayResource

---

**Purpose** Plays formatted sound data read from a resource or file.

**Prototype** Err SndPlayResource (SndPtr sound,  
Int32 ampScale, UInt32 flags)

**Parameters**

-> sound	A pointer to the beginning of the formatted sound (header and all). Currently, only WAVE data is recognized (see <a href="#">Comments</a> , below); in this case, sound must point to the "RIFF" ID (byte 0 in a simple .wav file).
-> ampScale	Amplitude scalar, in the range [0, 32k]. See <a href="#">SndStreamSetVolume</a> for information on how amplitude scalar values are applied.
-> flags	Settings flags. Currently, the only setting is function synchronization: Choose between

`sndFlagSync` and `sndFlagAsync`. The former tells the function to wait until all sound data has been fed to the DAC before returning (i.e. the function will return just a bit before the sound has finished playing). The `sndFlagAsync` flag tells the function to return immediately while playback continues in a separate thread.

As a convenience, the `sndFlagNormal` value is a shorthand for the set of “normal” flag settings. Currently, this is set to `sndFlagSync`.

<b>Result</b>		
	<code>errNone</code>	Success.
	<code>sndErrBadParam</code>	sound contains no data.
	<code>sndErrFormat</code>	The data is in an unsupported format.
	<code>sndErrMemory</code>	The function couldn't allocate sufficient memory.
	<i>other errors</i>	The device couldn't allocate system resources for the sound.

**Comments** Supported WAVE parameters are:

- uncompressed (PCM) or IMA 4-bit adaptive differential (IMA ADPCM). The ADPCM type is also known as DVI ADPCM; in a WAVE file, it's known as format 0x11.
- One or two-channels
- All normal sampling rates (8k, 11k, 22.05k, 44.1k, 48, 96k).

You can't interrupt or abort a resource playback once it's been initiated. The resource always play to the end of the data.

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.

**New SndStreamCreate**

**Purpose** Creates a new audio data stream that can be used to record or playback uncompressed, sampled audio data.

**Prototype**

```
Err SndStreamCreate (SndStreamRef *stream,
                     SndStreamMode mode, UInt32 sampleRate,
                     SndSampleType type, SndStreamWidth width,
                     SndStreamBufferCallback callback,
                     void *callbackArg, UInt32 bufferSize,
                     Boolean callbackIsARM)
```

<b>Parameters</b>	<code>&lt;- stream</code>	Token that represents the newly created stream.
	<code>-&gt; mode</code>	Constant that represents the “direction” of the data. Either <code>sndInput</code> (for recording), or <code>sndOutput</code> (for playback).
	<code>-&gt; sampleRate</code>	Sampling rate, in frames-per-second. The value passed here is the native rate of the data, given as a number (22050, 44100, 48000, etc.). The maximum rate is 96000.
	<code>-&gt; type</code>	Sample quantization and endianness (but see the section on “ <a href="#">Data Formats</a> ,” below).
	<code>-&gt; width</code>	A constant that represents the number of channels of data in the stream; either <code>sndMono</code> or <code>sndStereo</code> .
	<code>-&gt; callback</code>	A callback function that gets called when another buffer of data is needed.
	<code>-&gt; callbackArg</code>	Caller-defined data that gets passed to <code>callback</code> .
	<code>-&gt; bufferSize</code>	Preferred size (in frames) for the buffers that are passed to <code>callback</code> . Note that the actual size may be different.

-> callbackIsARM  
(68k only) Pass true if the callback function is written in ARM-native code; if it's 68k, pass false.

<b>Result</b>	errNone	Success.
	sndErrBadParam	stream is invalid, bufferFunc is NULL, the sampleRate is too high (greater than 96000), or the device doesn't like some other sound parameter value.
	sndErrorMemory	All streams are being used (there is a maximum of 16), or memory for this stream couldn't otherwise be allocated.
	<i>other errors</i>	The device couldn't allocate system resources for the stream.

**Comments** This function creates a new audio stream into which you can write (playback) or from which you can read (record) buffers of uncompressed, sampled audio data. The stream's "direction"—whether it will be used for recording or playback—is described by the mode argument.

You can create one input stream and as many as 15 output streams. The "active" end of a stream is hardwired to read from or write to the device's sound driver. This means you can't "redirect" an input stream to read from a file (for example), nor can you connect one output stream to another output stream in an attempt to create a filter chain. You can, however, collect data from the input stream, manipulate it, and then write it to an output stream.

### Data Formats

The format of the data that flows through the stream is described by the sampleRate, type, and width arguments:

- The data format that you specify for an input stream must match the data that's produced by the audio hardware.

## Sound Manager

### *Sampled Sound Functions*

---

- For an output stream, you can specify any of the formats that the Sound Manager supports; the data is automatically converted to the output hardware's native audio format. Whether your stream's format setting actually affects the hardware is undefined. For example, if you set an output stream to use a 48k sampling rate, that doesn't mean that the DAC will be set to 48k.

If you look at the [SndSampleTypeTag](#) constants, you'll see three flavors for each quantization type: There's a big-endian version, a little-endian version, and a native-endian version (defined as one of the other two). In general, you should use the native-endian version when choosing a value for the `type` parameter. The one exception to this (in Palm OS 5) is if your application is written for 68k, but uses an ARM callback function (i.e `callbackIsARM == true`). In this case you should use one of the little-endian formats as the `type` value when you create your stream.

### **Running the Stream**

The new stream starts running when you pass the stream token returned by this function to the [SndStreamStart](#) function. This initiates a series of calls to your callback function, which is where the action is: Each callback invocation is passed a buffer into which you write or from which you read a chunk of audio data. The `callback` function is also passed the `callbackArg` that you supply here. See [SndStreamBufferCallback](#) for more information on the callback function.

### **Buffering and Latency**

Currently, audio streams are double-buffered. With regard to playback, this means that while one buffer (buffer A) is being played, your callback function is placing data in the other buffer (B). When A is "empty," the Sound Manager seamlessly starts playing buffer B, and passes buffer A back to your callback; when B is empty, the Manager starts playing A, and passes back B, and so on. It's important that your callback function fills the data buffers as quickly as possible—certainly no longer than it takes to play a buffer of data. This same double-buffer scheme is also applied to sound recording although, of course, for recording you're emptying each buffer (and doing something with the data) in your callback function.

Regarding latency, you can use the bufferSize argument to suggest a buffer size and thereby increase or decrease latency, but you can't change the number of buffers. Keep in mind that the actual buffer size that's used may not be the same as the size you suggest; hardware and memory limitations may enforce a maximum or minimum buffer size. Also keep in mind that the bufferSize is measures in frames (not bytes).

[SndStreamStart](#), [SndStreamDelete](#),  
[SndStreamBufferCallback](#)

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.



---

## SndStreamDelete

**Purpose** Stops the stream and destroys it.

**Prototype** Err SndStreamDelete (SndStreamRef stream)

**Parameters** -> stream Stream token, as returned through  
[SndStreamCreate](#).

**Result** errNone Success.

  sndErrBadParam  
    stream is invalid.

**Comments** [SndStreamStop](#) is called before the stream is destroyed. You should never call this function as part of the implementation of a callback function.

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.

## Sound Manager

### Sampled Sound Functions

---



**New**

## SndStreamGetPan

**Purpose** Retrieves a stream's stereo balance.

**Prototype** Err SndStreamGetPan (SndStreamRef stream,  
Int32 \*pan)

**Parameters** -> stream Stream token, as returned through  
SndCreateRawStream.  
<- pan Pan value in the range [-1024 (hard left), 1024  
(hard right)]. Center balance is 0.

**Result** errNone Success.

  sndErrBadParam Invalid stream, or pan is NULL.

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.

**See Also** [SndStreamSetPan](#)



**New**

## SndStreamGetVolume

**Purpose** Retrieves the amplitude scalar for a sound stream.

**Prototype** Err SndStreamGetVolume (SndStreamRef stream,  
Int32 \*ampScale)

**Parameters** -> stream Stream token, as returned through  
SndCreateRawStream.  
<- ampScale Amplitude scalar, in the range [0, 32k]. See  
[SndStreamSetVolume](#) for more information.

**Result** errNone Success.

`sndErrBadParam`  
Invalid stream, or volume is NULL.

**See Also** [SndStreamSetVolume](#)

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.



## SndStreamPause

**Purpose** Pauses and resumes a sample stream.

**Prototype** Err SndStreamPause (SndStreamRef stream,  
Boolean pause)

**Parameters** -> stream Stream token, as returned by  
[SndStreamCreate](#).  
-> pause If true, the function pauses the stream; if it's  
false, it resumes the stream

**Result** errNone Success. Note that `errNone` is returned even if  
the stream is already in the requested state.  
`sndErrBadParam`  
Invalid stream.

**Comments** Currently, `SndStreamPause` is implemented through calls to  
[SndStreamStop](#) and [SndStreamStart](#) (the former if  
`pause==true`; the latter if `pause==false`). See those functions  
for details about “pausing” and “resuming” a sound stream.  
You can't nest pauses; a single resume request is effective,  
regardless of the number of times the stream has been told to pause.

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.

## Sound Manager

### *Sampled Sound Functions*

---



#### New SndStreamSetPan

**Purpose** Sets a stream's stereo balance.

**Prototype** Err SndStreamSetPan (SndStreamRef stream,  
Int32 pan);

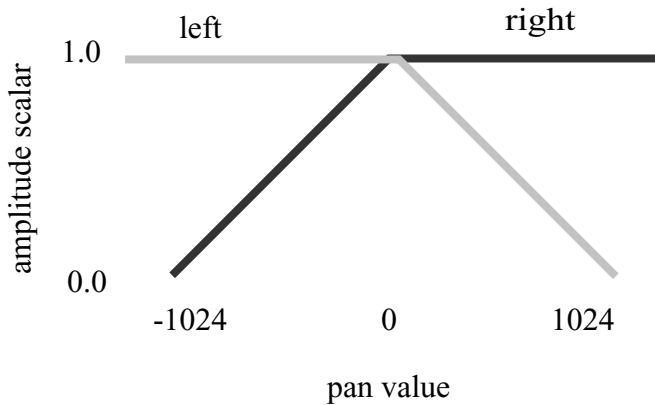
**Parameters**

-> stream	Stream token, as returned through SndCreateRawStream.
-> pan	Pan value in the range [-1024 (full left), 1024 (full right)]. Center balance is 0. As a convenience, you can use the values described in " <a href="#">Stereo Pan Constants</a> ." Note that values outside of the valid range may yield unexpected results (but don't generate an error).

**Result** errNone Success.

  sndErrBadParam  
                Invalid stream.

**Comments** The pan value is used as a scalar on a channel's volume such that a channel increases from 0 (inaudible) to full volume as the pan value moves from an extreme to 0. Graphically, it looks like this:



**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.

**See Also** [SndStreamGetPan](#)

  
**New**

## SndStreamSetVolume

**Purpose** Sets the amplitude scalar for a sound stream.

**Prototype** Err SndStreamSetVolume (SndStreamRef stream,  
Int32 ampScale);

**Parameters** -> stream Stream token, as returned through  
SndCreateRawStream.  
-> ampScale Amplitude scalar in the range [0, 32k]. Values  
less than 0 are converted to 1024 (unity gain).

**Result** errNone Success.

  sndErrBadParam Invalid stream.

**Comments** The ampScale value is applied as an amplitude scalar on the  
samples that this stream's callback function produces. The scalar is

## Sound Manager

### Sampled Sound Functions

---

in the range [0, 32k], where 1024 is unity gain (i.e. the samples are multiplied by 1.0). The mapping of ampScale to scalar is linear; thus a volume of 512 scales the samples by ~.5, 2048 scales by ~2.0, and so on.

To specify a user preference volume setting, use one of `sndSystemVolume`, `sndGameVolume`, or `sndAlarmVolume`. These values are guaranteed to be less than unity gain.

If the stream is stereo, both channels are scaled by the same amplitude scalar. To adjust the balance between the channels, use [SndStreamSetPan](#).

<b>Compatibility</b>	Implemented if <a href="#"><u>Sound Stream Feature Set</u></a> is present.
----------------------	--



## SndStreamStart

<b>Purpose</b>	Starts a sample stream running.
----------------	---------------------------------

<b>Prototype</b>	<code>Err SndStreamStart (SndStreamRef stream);</code>
------------------	--

<b>Parameters</b>	<code>-&gt; stream</code>	Stream token, as returned by <a href="#"><u>SndStreamCreate</u></a> .
-------------------	---------------------------	---

<b>Result</b>	<code>errNone</code>	Success. Note that <code>errNone</code> is returned even if the stream is already running.
	<code>sndErrBadParam</code>	<code>stream</code> is invalid.

<b>Comments</b>	If the stream is already running, the function returns immediately (with <code>errNone</code> ). If it isn't running, the function starts the stream by initiating invocations of its callback function. If it's paused (through <a href="#"><u>SndStreamPause</u></a> ), the stream is resumed.
-----------------	--

You can call this function from some other stream's callback function. In other words, a stream can tell another stream to start playing.

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.



## New **SndStreamStop**

**Purpose** Stops a sample stream running.

**Prototype** Err SndStreamStop (SndStreamRef stream);

**Parameters** -> stream Stream token, as returned by [SndStreamCreate](#).

**Result** errNone Success. Note that errNone is returned even if the stream has already been stopped.

  sndErrBadParam  
                stream is invalid.

**Comments** Stops a running sound stream by neglecting to call its callback function. The stream remains in this suspended state until you call [SndStreamStart](#).

You can call this function from the stream's own callback function. In other words, a stream can stop itself.

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.

# Sampled Sound Application-Defined Functions

**New**

## SndStreamBufferCallback

**Purpose** Used to deliver a data buffer from/into which you read/write sound data.

**Prototype** Err SndStreamBufferCallback (void \*userData,  
SndStreamRef stream, void \*buffer,  
UInt32 frameCount);

**Parameters**

-> userData	Caller-defined data, as provided by the callbackArg parameter to <a href="#">SndStreamCreate</a> .
-> stream	Token that represents the stream that this buffer belongs to.
-> buffer	The data buffer.
-> frameCount	Number of sample frames the buffer contains.

**Result** Currently, the return value is ignored.

**Comments** If this is an input (recording) stream, you read the data in buffer. If this is an output (playback) stream, you write data into buffer. In either case, you want to do this as quickly as possible to avoid data underflow.

Note that the function doesn't tell you anything about the format of the data. You can use the userData argument to pass this information into the function.

The callback function is executed in a task that's created and managed by the Sound Manager. Because of this, the function doesn't have access to the symbols that you create in your application. Again, use the userData argument if you need to pass pointers to your symbols.

**Compatibility** Implemented if [Sound Stream Feature Set](#) is present.

**See Also** [SndStreamCreate](#)

## **Sound Manager**

*Sampled Sound Application-Defined Functions*

---

# Standard IO

---

This chapter provides reference material for the standard IO API:

- [Standard IO Functions](#)
- [Standard IO Provider Functions](#)
- [Application-Defined Function](#)

The header files `StdIOPalm.h` and `StdIOProvider.h` declare the standard IO API. For more information on using the standard IO API, see the chapter "[Standard IO Applications](#)" in the *Palm OS Programmer's Companion*, vol. I.

## Standard IO Functions

The macros and functions in this section enable standard IO.

### fgetc

**Purpose** Macro that calls [`Siofgetc`](#) to return the next character from the input stream.

**Declared In** `StdIOPalm.h`

**Prototype** `fgetc (fs)`

**Parameters** `-> fs` An input stream from which to read the next character. You can specify only the value `stdin` for this parameter; alternate streams are not currently implemented.

**Result** The next character from the input stream. The return value `EOF` indicates an error occurred.

## Standard IO

### *Standard IO Functions*

---

## fgets

**Purpose** Macro that calls [Siofgets](#) to return a string from the input stream.

**Declared In** StdIOPalm.h

**Prototype** fgets (strP, maxChars, fs)

**Parameters**

<- strP	A pointer to the returned string.
-> maxChars	The number of characters to read from the input stream, plus one for the null terminator.
-> fs	An input stream from which to read the next character. You can specify only the value <code>stdin</code> for this parameter; alternate streams are not currently implemented.

**Result** A pointer to the string read from the input stream. If an error or EOF occurs before any characters are read, returns NULL.

**Comments** The returned string is always terminated by a null character.

## fprintf

**Purpose** Macro that calls [Siofprintf](#) to write formatted output to an output stream.

**Declared In** StdIOPalm.h

**Prototype** fprintf (fs, formatP, ...)

**Parameters**

-> fs	An output stream to which to write the formatted output. You can specify only the value <code>stdout</code> for this parameter; alternate streams are not currently implemented.
-> formatP	A pointer to a format string that controls how subsequent arguments are converted for output.

-> ...      Zero or more parameters to be formatted as specified by the `formatP` string.

**Result**      Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

**Comments**      This function internally calls [`StrVPrintF`](#) to do the formatting. See that function for details on which format specifications are supported.

## **fputc**

**Purpose**      Macro that calls [`Siofputc`](#) to write a character to the output stream.

**Declared In**      `StdIOPalm.h`

**Prototype**      `fputc (c, fs)`

**Parameters**      -> `c`      A character to write to the output stream.

                      -> `fs`      An output stream to which to write the character. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

**Result**      The character that was written. If an error occurs, the value `EOF` is returned.

## Standard IO

### *Standard IO Functions*

---

## fputs

**Purpose** Macro that calls [Siofputs](#) to write a string to the output stream.

**Declared In** StdIOPalm.h

**Prototype** fputs (strP, fs)

**Parameters**

-> strP	A pointer to the string to write.
-> fs	An output stream to which to write the string. You can specify only the value <code>stdout</code> for this parameter; alternate streams are not currently implemented.

**Result** Returns 0 on success and the value EOF on error.

## getchar

**Purpose** Macro that calls [Siofgetc](#) to read the next character from the `stdin` input stream.

**Declared In** StdIOPalm.h

**Prototype** getchar ()

**Result** The next character from the input stream. The return value EOF indicates an error occurred.

## gets

<b>Purpose</b>	Macro that calls <a href="#">Siogets</a> to read a string from the stdin input stream.
<b>Declared In</b>	StdIOPalm.h
<b>Prototype</b>	gets (strP)
<b>Parameters</b>	<- strP                    A pointer to the returned string.
<b>Result</b>	A pointer to the string read from the input stream. If an error or EOF occurs before any characters are read, returns NULL.
<b>Comments</b>	The returned string does not include a null terminator. You must ensure that the input line, if any, is sufficiently short to fit in the string.

## printf

<b>Purpose</b>	Macro that calls <a href="#">Sioprintf</a> to write formatted output to the stdout output stream.
<b>Declared In</b>	StdIOPalm.h
<b>Prototype</b>	printf (formatP, ...)
<b>Parameters</b>	-> formatP                    A pointer to a format string that controls how subsequent arguments are converted for output. -> ...                        Zero or more parameters to be formatted as specified by the formatP string.
<b>Result</b>	Returns the number of characters written out (not including the null terminator used to end output strings).

## Standard IO

### Standard IO Functions

---

**Comments** This function internally calls [StrVPrintF](#) to do the formatting. See that function for details on which format specifications are supported. Returns a negative number if there is an error.

## putc

**Purpose** Macro that calls [Siofputc](#) to write a character to the output stream.

**Declared In** StdIOPalm.h

**Prototype** putc (c, fs)

**Parameters** -> c A character to write to the output stream.  
-> fs An output stream to which to write the character. You can specify only the value `stdout` for this parameter; alternate streams are not currently implemented.

**Result** The character that was written. If an error occurs, the value `EOF` is returned.

## putchar

**Purpose** Macro that calls [Siofputc](#) to write a character to the `stdout` output stream.

**Declared In** StdIOPalm.h

**Prototype** putchar (c)

**Parameters** -> c A character to write to the `stdout` output stream.

**Result** The character that was written. If an error occurs, the value `EOF` is returned.

**puts**

**Purpose** Macro that calls [\\_Sioputs](#) to write a string to the output stream stdout.

## Declared In StdIOPalm.h

**Prototype** puts (strP)

**Parameters**    `-> strP`              A pointer to the string to write to stdout.

**Result** Returns a nonnegative value on success and the value EOF on error.

## SioAddCommand

**Purpose** Adds a built-in command that is supplied by the standard IO provider application.

## **Declared In** StdIOPalm.h

**Prototype** void SioAddCommand (const Char \*cmdStr,  
SioMainProcPtr cmdProcP)

<b>Parameters</b>	-> cmdStr	Pointer to a string that is the command name.
	-> cmdProcP	Pointer to the command entry point function (the <a href="#">SioMain</a> function).

**Result** Returns nothing.

<b>Comments</b>	This routine is useful for registering a command that is inside the standard IO provider application instead of in its own database.  This routine must be used to test commands under the Simulator since it can't launch application databases.
-----------------	---

## Standard IO

### Standard IO Functions

---

## Siofgetc

Return the next character from the input stream.

**Declared In** StdIOPalm.h

**Prototype** Int16 Siofgetc (FILE \*fs)

**Parameters** -> fs An input stream from which to read the next character. You can specify only the value `stdin` for this parameter; alternate streams are not currently implemented.

**Result** The next character from the input stream. The return value `EOF` indicates an error occurred.

**See Also** [fgetc](#)

## Siofgets

**Purpose** Return a string from the input stream.

**Declared In** StdIOPalm.h

**Prototype** Char \*Siofgets (Char \*strP, UInt16 maxChars, FILE \*fs)

**Parameters** <- strP A pointer to the returned string.  
-> maxChars The number of characters to read from the input stream, plus one for the null terminator.  
-> fs An input stream from which to read the next character. You can specify only the value `stdin` for this parameter; alternate streams are not currently implemented.

**Result** A pointer to the string read from the input stream. If an error or `EOF` occurs before any characters are read, returns `NULL`.

**Comments** The returned string is always terminated by a null character.

**See Also** [fgets](#)

## Siofprintf

**Purpose** Write formatted output to an output stream.

**Declared In** StdIOPalm.h

**Prototype** Int16 Siofprintf (FILE \*fs, const Char \*formatP,  
...)

**Parameters**

-> fs	An output stream to which to write the formatted output. You can specify only the value <code>stdout</code> for this parameter; alternate streams are not currently implemented.
-> formatP	A pointer to a format string that controls how subsequent arguments are converted for output.
-> ...	Zero or more parameters to be formatted as specified by the <code>formatP</code> string.

**Result** Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

**Comments** This function internally calls [StrVPrintF](#) to do the formatting. See that function for details on which format specifications are supported.

**See Also** [fprintf](#)

## Standard IO

### *Standard IO Functions*

---

## Siofputc

**Purpose** Write a character to the output stream.

**Declared In** StdIOPalm.h

**Prototype** Int16 Siofputc (Int16 c, FILE \*fs)

**Parameters**

-> c	A character to write to the output stream.
-> fs	An output stream to which to write the character. You can specify only the value <code>stdout</code> for this parameter; alternate streams are not currently implemented.

**Result** The character that was written. If an error occurs, the value `EOF` is returned.

**See Also** [fputc](#)

## Siofputs

**Purpose** Write a string to the output stream.

**Declared In** StdIOPalm.h

**Prototype** Int16 Siofputs (const Char \*strP, FILE \*fs)

**Parameters**

-> strP	A pointer to the string to write.
-> fs	An output stream to which to write the string. You can specify only the value <code>stdout</code> for this parameter; alternate streams are not currently implemented.

**Result** Returns 0 on success and the value `EOF` on error.

**See Also** [fputs](#)

## Siogets

<b>Purpose</b>	Read a string from the <code>stdin</code> input stream.
<b>Declared In</b>	<code>StdIOPalm.h</code>
<b>Prototype</b>	<code>Char *Siogets (Char *strP)</code>
<b>Parameters</b>	<code>&lt;- strP</code> A pointer to the returned string.
<b>Result</b>	A pointer to the string read from the input stream. If an error or EOF occurs before any characters are read, returns <code>NULL</code> .
<b>Comments</b>	The returned string does not include a null terminator. You must ensure that the input line, if any, is sufficiently short to fit in the string.
<b>See Also</b>	<a href="#">gets</a>

## Sioprintf

<b>Purpose</b>	Write formatted output to the <code>stdout</code> output stream.
<b>Declared In</b>	<code>StdIOPalm.h</code>
<b>Prototype</b>	<code>Int16 Sioprintf (const Char *formatP, ...)</code>
<b>Parameters</b>	<code>-&gt; formatP</code> A pointer to a format string that controls how subsequent arguments are converted for output. <code>-&gt; ...</code> Zero or more parameters to be formatted as specified by the <code>formatP</code> string.
<b>Result</b>	Returns the number of characters written out (not including the null terminator used to end output strings).

## Standard IO

### *Standard IO Functions*

---

**Comments** This function internally calls [StrVPrintF](#) to do the formatting. See that function for details on which format specifications are supported. Returns a negative number if there is an error.

**See Also** [printf](#)

## Sioputs

**Purpose** Write a string to the output stream stdout.

**Declared In** StdIOPalm.h

**Prototype** Int16 Sioputs (const Char \*strP)

**Parameters** -> strP A pointer to the string to write to stdout.

**Result** Returns a nonnegative value on success and the value EOF on error.

**See Also** [puts](#)

## Siosystem

**Purpose** Execute another Stdio command.

**Declared In** StdIOPalm.h

**Prototype** Int16 Siosystem (const Char \*cmdStrP)

**Parameters** -> cmdStrP A pointer to a string containing the command line to execute.

**Result** Returns a value  $\geq 0$  on success or  $< 0$  on failure.

**Comments** This function first looks for a built-in command with the specified name. If none is found, it looks for a Stdio application database with

the name "Cmd-*cmdname*" where *cmdname* is the first word in the command string cmdStrP.

**See Also** [SioExecCommand](#), [system](#)

## Siovfprintf

**Purpose** Write formatted output to the `stdout` output stream.

**Declared In** `StdIOPalm.h`

**Prototype** `Int16 Siovfprintf (FILE *fs, const Char *formatP,  
_Palm_va_list args)`

**Parameters**

-> <code>fs</code>	An output stream to which to write the formatted output. You can specify only the value <code>stdout</code> for this parameter; alternate streams are not currently implemented.
-> <code>formatP</code>	A pointer to a format string that controls how subsequent arguments are converted for output.
-> <code>args</code>	A pointer to a list of zero or more parameters to be formatted as specified by the <code>formatP</code> string.

**Result** Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

**Comments** This function internally calls [StrVPrintF](#) to do the formatting. See that function for details on which format specifications are supported.

**See Also** [vfprintf](#)

## Standard IO

### Standard IO Functions

---

## sprintf

<b>Purpose</b>	Macro that calls <a href="#">StrPrintF</a> to write formatted output to the stdout output stream.				
<b>Declared In</b>	StdIOPalm.h				
<b>Prototype</b>	<code>sprintf (formatP, ...)</code>				
<b>Parameters</b>	<table><tr><td><code>-&gt; formatP</code></td><td>A pointer to a format string that controls how subsequent arguments are converted for output.</td></tr><tr><td><code>-&gt; ...</code></td><td>Zero or more parameters to be formatted as specified by the <code>formatP</code> string.</td></tr></table>	<code>-&gt; formatP</code>	A pointer to a format string that controls how subsequent arguments are converted for output.	<code>-&gt; ...</code>	Zero or more parameters to be formatted as specified by the <code>formatP</code> string.
<code>-&gt; formatP</code>	A pointer to a format string that controls how subsequent arguments are converted for output.				
<code>-&gt; ...</code>	Zero or more parameters to be formatted as specified by the <code>formatP</code> string.				
<b>Result</b>	Returns the number of characters written out (not including the null terminator used to end output strings).				
<b>Comments</b>	See <a href="#">StrVPrintF</a> for details on which format specifications are supported. Returns a negative number if there is an error.				

## system

<b>Purpose</b>	Macro that calls <a href="#">Siosystem</a> to execute another Stdio command.		
<b>Declared In</b>	StdIOPalm.h		
<b>Prototype</b>	<code>system (cmdStrP)</code>		
<b>Parameters</b>	<table><tr><td><code>-&gt; cmdStrP</code></td><td>A pointer to a string containing the command line to execute.</td></tr></table>	<code>-&gt; cmdStrP</code>	A pointer to a string containing the command line to execute.
<code>-&gt; cmdStrP</code>	A pointer to a string containing the command line to execute.		
<b>Result</b>	Returns a value $\geq 0$ on success or $< 0$ on failure.		
<b>Comments</b>	This function first looks for a built-in command with the specified name. If none is found, it looks for a Stdio application database with		

the name "Cmd-*cmdname*" where *cmdname* is the first word in the command string cmdStrP.

**See Also** [SioExecCommand](#)

## **vfprintf**

**Purpose** Macro that calls [Siovfprintf](#) to write formatted output to the stdout output stream.

**Declared In** StdIOPalm.h

**Prototype** vfprintf (fs, formatP, args)

**Parameters**

-> fs	An output stream to which to write the formatted output. You can specify only the value <code>stdout</code> for this parameter; alternate streams are not currently implemented.
-> formatP	A pointer to a format string that controls how subsequent arguments are converted for output.
-> args	A pointer to a list of zero or more parameters to be formatted as specified by the <code>formatP</code> string.

**Result** Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

**Comments** This function internally calls [StrVPrintF](#) to do the formatting. See that function for details on which format specifications are supported.

## Standard IO

### *Standard IO Provider Functions*

---

## **vsprintf**

**Purpose** Macro that calls [StrVPrintF](#) to write formatted output to the stdout output stream.

**Declared In** StdIOPalm.h

**Prototype** vsprintf (fs, formatP, args)

<b>Parameters</b>	-> fs	An output stream to which to write the formatted output. You can specify only the value <code>stdout</code> for this parameter; alternate streams are not currently implemented.
	-> formatP	A pointer to a format string that controls how subsequent arguments are converted for output.
	-> args	A pointer to a list of zero or more parameters to be formatted as specified by the y string.

**Result** Returns the number of characters written out (not including the null terminator used to end output strings). Returns a negative number if there is an error.

**Comments** See [StrVPrintF](#) for details on which format specifications are supported.

## Standard IO Provider Functions

These functions are used by a standard IO provider application.

## SioClearScreen

**Purpose** Clears the entire standard IO output field.

**Declared In** StdIOProvider.h

**Prototype** void SioClearScreen (void)

**Parameters** None.

**Result** Returns nothing.

## SioExecCommand

**Purpose** Executes a command line.

**Declared In** StdIOProvider.h

**Prototype** Int16 SioExecCommand (const Char \*cmd)

**Parameters** -> cmd A pointer to a string containing the command line to execute.

**Result** Returns a value  $\geq 0$  on success or  $< 0$  on failure.

**Comments** This function first looks for a built-in command with the specified name. If none is found, it looks for a Stdio application database with the name "Cmd-*cmdname*" where *cmdname* is the first word in the command string cmd.

If you pass the string "help" or "?" for the cmd parameter, SioExecCommand causes a help string to be printed for each built-in command. It actually executes each built-in command, passing the string "?" as argv [1]. Each command should handle this argument by printing a help line.

The SioExecCommand function is faster than calling [system](#) to execute a command. However, SioExecCommand can be called

## **Standard IO**

### *Standard IO Provider Functions*

---

only by the standard IO provider application, not the standard IO application.

## **SioFree**

**Purpose** Closes down the standard IO manager.

**Declared In** StdIOProvider.h

**Prototype** Err SioFree (void)

**Parameters** None.

**Result** Returns 0 on success.

## **SioHandleEvent**

**Purpose** Handles an event in the form that contains the standard IO output field and scroll arrows if the event belongs to the text field or scroll arrows.

**Declared In** StdIOProvider.h

**Prototype** Boolean SioHandleEvent (SysEventType \*event)

**Parameters** -> event Pointer to an EventType structure.

**Result** Returns true if the event was handled and should not be processed by the application's own form event handler; returns false otherwise.

**Comments** This function must be called from the form event handler before it does its own processing with any of the objects unrelated to standard IO in the form.

## SioInit

**Purpose** Initializes the standard IO manager.

**Declared In** StdIOProvider.h

**Prototype** Err SioInit (UInt16 formID, UInt16 fieldID,  
                  UInt16 scrollerID)

**Parameters**

-> formID	The ID of the form that contains the input/output field.
-> fieldID	The ID of the field to be used for input/output.
-> scrollerID	The ID of the scroller associated with the input/output form.

**Result** Returns 0 on success.

## **Standard IO**

*Application-Defined Function*

---

# **Application-Defined Function**

You must supply this function in your stdio application.

## **SioMain**

**Purpose** The main entry point for the stdio application.

**Declared In** StdIOPalm.h

**Prototype** Int16 SioMain (UInt16 argc, const Char \*argv[] )

**Parameters** -> argc The number of parameters passed on the command line.  
-> argv An array of character pointers, one for each parameter passed on the command line.

**Result** The return value from this routine is passed back to the system call that invoked it. Return 0 for no error.

# String Manager

---

This chapter provides reference material for the string manager. The string manager API is declared in the header file `StringMgr.h`.

For more information, see [Chapter 8, “Text,”](#) on page 251 of the *Palm OS Programmer’s Companion*, vol. I.

## String Manager Functions

### StrAToI

**Purpose** Convert a string to an integer.

**Declared In** `StringMgr.h`

**Prototype** `Int32 StrAToI (const Char *str)`

**Parameters** `-> str` Pointer to a string to convert.

**Result** Returns the integer.

**Comments** Use this function instead of the standard `atoi` routine.

## String Manager

### *String Manager Functions*

---

## StrCaselessCompare

**Purpose** Compare two strings with case and accent insensitivity.

**Declared In** StringMgr.h

**Prototype** Int16 StrCaselessCompare (const Char \*s1,  
const Char \*s2)

**Parameters**

-> s1	Pointer to a string.
-> s2	Pointer to a string.

**Result** Returns 0 if the strings match.

Returns a positive number if s1 > s2.

Returns a negative number if s1 < s2.

**Comments** Use this function instead of the standard `strcmp` routine.

To support systems that use multi-byte character encodings, consider using [TxtCaselessCompare](#) instead of this function (or [TxtCompare](#) for a case-sensitive comparison). Both functions can match single-byte characters with their multi-byte equivalents, but `TxtCaselessCompare` can also return the length of the matching text.

**See Also** [StrNCaselessCompare](#), [TxtCaselessCompare](#),  
[StrCompare](#), [StrNCompare](#)

## StrCat

<b>Purpose</b>	Concatenate one null-terminated string to another.
<b>Declared In</b>	StringMgr.h
<b>Prototype</b>	Char *StrCat (Char *dst, const Char *src)
<b>Parameters</b>	-> dst                    Pointer to the null-terminated destination string. -> src                    Pointer to the null-terminated source string.
<b>Result</b>	Returns a pointer to the destination string.
<b>Comments</b>	Use this function instead of the standard <code>strcat</code> routine.

## StrChr

<b>Purpose</b>	Look for a character within a string.
<b>Declared In</b>	StringMgr.h
<b>Prototype</b>	Char *StrChr (const Char *str, WChar chr)
<b>Parameters</b>	-> str                    Pointer to the string to be searched. -> chr                    Character to search for.
<b>Result</b>	Returns a pointer to the first occurrence of character in <code>str</code> . Returns NULL if the character is not found.
<b>Comments</b>	Use this function instead of the standard <code>strchr</code> routine.  This function can handle both single-byte characters and multi-byte characters correctly. However, you should make sure that you pass a <code>WChar</code> variable to <code>StrChr</code> instead of a <code>Char</code> . If you pass a <code>Char</code> variable, the function sign-extends the variable to a <code>WChar</code> , which causes problems if the value is 0x80 or higher.

## String Manager

### *String Manager Functions*

---

**Compatibility** This routine does not correctly find a '\0' character on Palm OS® version 1.0.

**See Also** [StrStr](#)

## StrCompare

**Purpose** Compare two strings.

**Declared In** StringMgr.h

**Prototype** Int16 StrCompare (const Char \*s1, const Char \*s2)

**Parameters** -> s1 Pointer to a string.  
-> s2 Pointer to a string.

**Result** Returns 0 if the strings match.

Returns a positive number if s1 sorts after s2 alphabetically.

Returns a negative number if s1 sorts before s2 alphabetically.

**Comments** Use this function or [StrCompareAscii](#) instead of the standard strcmp routine. This function is case sensitive.

To support systems that use multi-byte character encodings, consider using [TxtCompare](#) instead of this function. Both functions can match single-byte characters with their multi-byte equivalents, but TxtCompare can also return the length of the matching text.

**Compatibility** Prior to Palm OS 4.0, StrCompare and [TxtCompare](#) only performed one level of comparison and returned as soon as they found two unequal characters. For example, if you compared the string "celery" with the string "Cauliflower," both functions returned a value indicating that "celery" should appear before "Cauliflower" because they sorted "c" before "C."

In Palm OS 4.0, StrCompare calls TxtCompare, and TxtCompare performs a comparison using up to six comparison tables for sorting

with increasing precision. As a result, in Palm OS 4.0 and higher, StrCompare sorts “Cauliflower” before “celery.”

**See Also** [StrNCompare](#), [StrNCaselessCompare](#), [TxtCaselessCompare](#)

## StrCompareAscii

**Purpose** Compare two ASCII strings.

**Declared In** StringMgr.h

**Prototype** Int16 StrCompareAscii (const Char \*s1,  
const Char \*s2)

**Parameters** -> s1                    Pointer to a string.  
            -> s2                    Pointer to a string.

**Result** Returns 0 if the strings match.

Returns a positive number if s1 sorts after s2 alphabetically.

Returns a negative number if s1 sorts before s2 alphabetically.

**Comments** Use this function instead of the standard `strcmp` routine. Use it to do case-sensitive comparisons on strings that are guaranteed to be 7-bit ASCII strings. This function performs a fast, simple byte-to-byte comparison that is guaranteed never to change.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [StrCompare](#), [StrNCompare](#), [TxtCompare](#),  
[StrCaselessCompare](#), [StrNCaselessCompare](#),  
[TxtCaselessCompare](#), [StrNCompareAscii](#)

## **String Manager**

### *String Manager Functions*

---

## **StrCopy**

**Purpose** Copy one string to another.

**Declared In** StringMgr.h

**Prototype** Char \*StrCopy (Char \*dst, const Char \*src)

**Parameters** -> dst Pointer to the destination string.  
-> src Pointer to the source string.

**Result** Returns a pointer to the destination string.

**Comments** Use this function instead of the standard `strcpy` routine.  
This function does not work properly with overlapping strings.

## **StrDelocalizeNumber**

**Purpose** Delocalize a number passed in as a string. Convert the number from any localized notation to US notation (decimal point and thousandth comma). The current thousand and decimal separators have to be passed in.

**Declared In** StringMgr.h

**Prototype** Char \*StrDelocalizeNumber (Char \*s,  
Char thousandSeparator, Char decimalSeparator)

**Parameters** <-> s Pointer to the number as an ASCII string.  
-> thousandSeparator Current thousand separator.  
-> decimalSeparator Current decimal separator.

**Result** Returns a pointer to the changed number and modifies the string in s.

**Comments** The current thousandSeparator and decimalSeparator can be determined by obtaining the value of the prefNumberFormat preference using [PrefGetPreference](#) and then passing the returned [NumberFormatType](#) to [LocGetNumberSeparators](#). For example:

```
Char *localizedNum;
NumberFormatType numFormat;
Char thousandsSeparator, decimalSeparator;

numFormat = (NumberFormatType)
    PrefGetPreference(prefNumberFormat);
LocGetNumberSeparators(numFormat,
    &thousandsSeparator, &decimalSeparator);
StrDelocalizeNumber(localizedNum,
    thousandsSeparator, decimalSeparator);
```

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

**See Also** [StrLocalizeNumber](#), [LocGetNumberSeparators](#)

## StrIToA

**Purpose** Convert an integer to ASCII.

**Declared In** StringMgr.h

**Prototype** Char \*StrIToA (Char \*s, Int32 i)

**Parameters** <- s Pointer to a string of size maxStrIToALen in which to store the results.  
-> i Integer to convert.

**Result** Returns a pointer to the result string.

**See Also** [StrAToI](#), [StrIToH](#)

## String Manager

### *String Manager Functions*

---

## StrIToH

**Purpose** Convert an integer to hexadecimal ASCII.

**Declared In** StringMgr.h

**Prototype** Char \*StrIToH (Char \*s, UInt32 i)

**Parameters** <- s                    Pointer to a string in which to store the results.  
      -> i                    Integer to convert.

**Result** Returns the string pointer s.

**See Also** [StrIToA](#)

## StrLen

**Purpose** Compute the length of a string.

**Declared In** StringMgr.h

**Prototype** UInt16 StrLen (const Char \*src)

**Parameters** >- src                    Pointer to a string.

**Result** Returns the length of the string in bytes.

**Comments** Use this function instead of the standard `strlen` routine.

This function returns the length of the string in bytes. On systems that support multi-byte characters, the number returned does not always equal the number of characters.

**Compatibility** In Palm OS 3.5 and Palm OS 4.x this function was declared to return an Int16. In Palm OS 5, and prior to Palm OS 3.5, this function returns a UInt16.

## StrLocalizeNumber

**Purpose** Convert a number (passed in as a string) to localized format, using a specified thousands separator and decimal separator.

**Declared In** StringMgr.h

**Prototype** Char \*StrLocalizeNumber (Char \*s,  
Char thousandSeparator, Char decimalSeparator)

## String Manager

### String Manager Functions

---

**Parameters**    <-> s                      Numeric ASCII string to localize.

                        -> thousandSeparator  
                                    Localized thousand separator.

                        -> decimalSeparator  
                                    Localized decimal separator.

**Result**    Returns a pointer to the changed number. Converts the number string in s by replacing all occurrences of "," with thousandSeparator and all occurrences of "." with decimalSeparator.

**Comments**    The current thousandSeparator and decimalSeparator can be determined by obtaining the value of the prefNumberFormat preference using [PrefGetPreference](#) and then passing the returned [NumberFormatType](#) to [LocGetNumberSeparators](#). For example:

```
Char *localizedNum;  
NumberFormatType numFormat;  
Char thousandsSeparator, decimalSeparator;  
  
numFormat = (NumberFormatType)  
    PrefGetPreference(prefNumberFormat);  
LocGetNumberSeparators(numFormat,  
    &thousandsSeparator, &decimalSeparator);  
StrLocalizeNumber(localizedNum,  
    thousandsSeparator, decimalSeparator);
```

**Compatibility**    Implemented only if [2.0 New Feature Set](#) is present.

**See Also**    [StrDelocalizeNumber](#)

## StrNCaselessCompare

<b>Purpose</b>	C.compares two strings out to <i>n</i> characters with case and accent insensitivity.
<b>Declared In</b>	StringMgr.h
<b>Prototype</b>	Int16 StrNCaselessCompare (const Char *s1, const Char *s2, Int32 n)
<b>Parameters</b>	-> s1 Pointer to the first string. -> s2 Pointer to the second string. -> n Length in bytes of the text to compare.
<b>Result</b>	Returns 0 if the strings match. Returns a positive number if s1 > s2. Returns a negative number if s1 < s2.
<b>Comments</b>	To support systems that use multi-byte character encodings, consider using <a href="#">TxtCaselessCompare</a> instead of this function (or <a href="#">TxtCompare</a> for a case-sensitive comparison). Both functions can match single-byte characters with their multi-byte equivalents, but TxtCaselessCompare can also return the length of the matching text.
<b>Compatibility</b>	Implemented only if <a href="#">2.0 New Feature Set</a> is present. As of Palm OS 4.0, both s1 and s2 must be null-terminated strings.
<b>See Also</b>	<a href="#">StrNCompare</a> , <a href="#">StrCaselessCompare</a> , <a href="#">TxtCaselessCompare</a> , <a href="#">StrCompare</a>

## String Manager

### *String Manager Functions*

---

## StrNCat

**Purpose** Concatenates one string to another clipping the destination string to a maximum of *n* bytes (including the null character at the end).

---

**IMPORTANT:** The Palm OS implementation of StrNCat differs from the implementation in the standard C library. See the Comments section for details.

---

**Declared In** StringMgr.h

**Prototype** Char \*StrNCat (Char \*dst, const Char \*src,  
Int16 n)

**Parameters**

-> dst	Pointer to the null-terminated destination string.
-> src	Pointer to the source string.
-> n	Maximum length in bytes for dst, including the terminating null character.

**Result** Returns a pointer to the destination string.

**Comment** This function differs from the standard C strncat function in these ways:

- StrNCat treats the parameter *n* as the maximum length in bytes for dst. That means it will copy at most *n* - StrLen(dst) - 1 bytes from src. The standard C function always copies *n* bytes from src into dst. (It copies the entire src into dst if the length of src is less than *n*).
- If the length of the destination string reaches *n* - 1, StrNCat stops copying bytes from src and appends the terminating null character to dst. If the length of the destination string is already greater than or equal to *n* - 1 before the copying begins, StrNCat does not copy any data from src.
- In the standard C function, if src is less than *n*, the entire src string is copied into dst and then the remaining space is filled with null characters. StrNCat does not fill the

remaining space with null characters in released ROMs. In debug ROMs, `StrNCat` fills the remaining bytes with the value `0xFE`.

On systems with multi-byte character encodings, this function makes sure that it does not copy part of a multi-byte character. If the last byte copied from `src` contains the high-order or middle byte of a multi-byte character, `StrNCat` backs up in `dst` until the byte after the end of the previous character, and replaces that byte with a null character.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## StrNCompare

**Purpose** Compare two strings out to  $n$  bytes. This function is case and accent sensitive.

**Declared In** `StringMgr.h`

**Prototype** `Int16 StrNCompare (const Char *s1,  
const Char *s2, Int32 n)`

**Parameters**

-> <code>s1</code>	Pointer to a string.
-> <code>s2</code>	Pointer to a string.
-> <code>n</code>	Length in bytes of text to compare.

**Result** Returns 0 if the strings match.

Returns a positive number if `s1 > s2`.

Returns a negative number if `s1 < s2`.

**Comments** To support systems that use multi-byte character encodings, consider using [TxtCompare](#) instead of this function. Both functions can match single-byte characters with their multi-byte equivalents, but `TxtCompare` can also return the length of the matching text.

## String Manager

### *String Manager Functions*

---

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present. As of Palm OS 4.0, both s1 and s2 must be null-terminated strings.

**See Also** [StrCompare](#), [StrNCaselessCompare](#), [StrCaselessCompare](#), [TxtCaselessCompare](#), [StrNCompareAscii](#)

## StrNCompareAscii

**Purpose** Compare two ASCII strings out to *n* bytes.

**Declared In** StringMgr.h

**Prototype** Int16 StrNCompareAscii (const Char \*s1,  
const Char \*s2, Int32 n)

**Parameters**

-> s1	Pointer to a string.
-> s2	Pointer to a string.
-> n	Length in bytes of text to compare.

**Result** Returns 0 if the strings match.

Returns a positive number if s1 sorts after s2 alphabetically.

Returns a negative number if s1 sorts before s2 alphabetically.

**Comments** Use this function instead of the standard strncmp routine. Use it to do case-sensitive comparisons on strings that are guaranteed to be 7-bit ASCII strings. This function performs a fast, simple byte-to-byte comparison that is guaranteed never to change.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [StrCompare](#), [StrNCompare](#), [TxtCompare](#),  
[StrCaselessCompare](#), [StrNCaselessCompare](#),  
[TxtCaselessCompare](#), [StrCompareAscii](#)

## StrNCopy

**Purpose** Copies up to  $n$  bytes from a source string to the destination string. Terminates `dst` string at index  $n-1$  if the source string length was  $n-1$  or less.

**Declared In** `StringMgr.h`

**Prototype** `Char *StrNCopy (Char *dst, const Char *src,  
Int16 n)`

**Parameters**

<code>-&gt; dst</code>	Pointer to the destination string.
<code>-&gt; src</code>	Pointer to the source string.
<code>-&gt; n</code>	Maximum number of bytes to copy from <code>src</code> string.

**Result** Returns nothing.

**Comments** On systems with multi-byte character encodings, this function makes sure that it does not copy part of a multi-byte character. If the  $n$ th byte of `src` contains the high-order or middle byte of a multi-byte character, `StrNCopy` backs up in `dst` until the byte after the end of the previous character, and replaces the remaining bytes (up to  $n-1$ ) with nulls.

Be aware that the  $n$ th byte of `dst` upon return may contain the last byte of a multi-byte character. If you plan to terminate the string by setting its last character to NULL, you must not pass the entire length of the string to `StrNCopy`. If you do, your code may overwrite the final byte of the last character.

```
// WRONG! You may overwrite part of multi-byte
// character.
Char dst [n];
StrNCopy(dst, src, n);
dst [n-1] = chrNull;
```

Instead, if you write to the last byte of the destination string, pass one less than the size of the string to `StrNCopy`.

## String Manager

### *String Manager Functions*

---

```
// RIGHT. Instead pass n-1 to StrNCopy.  
Char dst[n];  
StrNCopy(dst, src, n-1);  
dst[n-1] = chrNull;
```

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## StrPrintF

**Purpose** Implements a subset of the ANSI C `sprintf` call, which writes formatted output to a string.

**Declared In** `StringMgr.h`

**Prototype** `Int16 StrPrintF (Char *s,  
const Char *formatStr, ...)`

**Parameters**

-> s	Pointer to a string into which the results are written.
-> formatStr	Pointer to the format specification string.
...	Zero or more arguments to be formatted as specified by <code>formatStr</code> .

**Result** Number of characters written to destination string. Returns a negative number if there is an error.

**Comments** This function internally calls [StrVPrintF](#) to do the formatting. See that function for details on which format specifications are supported.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

**See Also** [StrVPrintF](#)

## StrStr

**Purpose** Look for a substring within a string.

**Declared In** StringMgr.h

**Prototype** Char \*StrStr (const Char \*str, const Char \*token)

**Parameters**

-> str	Pointer to the string to be searched.
-> token	Pointer to the string to search for.

**Result** Returns a pointer to the first occurrence of token in str or NULL if not found.

**Comments** Use this function instead of the standard `strstr` routine.

On systems with multi-byte character encodings, this function makes sure that it does not match only part of a multi-byte character. If the matching strings begins at an inter-character boundary, then this function returns NULL.

---

**NOTE:** If the value of the token parameter is the empty string, this function returns NULL. This is different than the standard `strstr` function, which returns str when token is the empty string.

---

**See Also** [StrChr](#)

## StrToLower

**Purpose** Convert all the characters in a string to lowercase.

**Declared In** StringMgr.h

**Prototype** Char \*StrToLower (Char \*dst, const Char \*src)

**Parameters**

-> dst	Pointer to a string.
--------	----------------------

## String Manager

### *String Manager Functions*

---

-> src                    Pointer to a null-terminated string.

**Result**    Returns a pointer to the destination string.

**Compatibility**    Prior to Palm OS version 3.5, this function only converted accented characters on Japanese devices. On Palm OS version 3.5 and higher, all characters are appropriately lowercased, including accented characters on Latin devices.

## StrVPrintF

**Purpose**    Implements a subset of the ANSI C `vsprintf` call, which writes formatted output to a string.

**Declared In**    `StringMgr.h`

**Prototype**    `Int16 StrVPrintF (Char *s, const Char *formatStr,  
va_list arg)`

**Parameters**    `<- s`                    Pointer to a string into which the results are written. This string is always terminated by a null terminator.

`-> formatStr`            Pointer to the format specification string.

`-> arg`                    Pointer to a list of zero or more parameters to be formatted as specified by the `formatStr` string.

**Result**    Number of characters written to destination string, not including the null terminator. Returns a negative number if there is an error.

**Comments**    Like the C `vsprintf` function, this function is designed to be called by your own function that takes a variable number of arguments and passes them to this function. For details on how to use it, see “[Using the StrVPrintF Function](#)” on page 267 of the *Palm OS Programmer’s Companion*, vol. I, or refer to `vsprintf` in a standard C reference book.

Currently, only the conversion specifications %d, %i, %u, %x, %s, and %c are implemented by `StrVPrintF` (and related functions). Optional modifiers that are supported include: +, -, <space>, \*, <digits>, h and l (long). Following is a brief description of how these format specifications work (see a C book for more details).

Each conversion specification begins with the % character. Following the % character, there may be one or more of the characters list in [Table 47.1](#), in sequence.

**Table 47.1 StrVPrintF Format Specification**

Character	Description
+	Specifies that a sign always be placed before a number produced by a signed conversion. A + overrides a space if both are used. Example: <code>StrPrintF(s, "%+d %+d", 4, -5);</code> Output to s: +4 -5
-	Specifies that the printed value is left justified within the field width allowed for it. Example: <code>StrPrintF(s, "%5d%-5d%d", 6, 9, 8);</code> Output to s: 69 8
<space>	Specifies that a minus sign always be placed before a negative number and a space before a positive number. Example: <code>StrPrintF(s, "% d % d", 4, -5);</code> Output to s: 4 -5

## String Manager

### *String Manager Functions*

---

**Table 47.1 StrVPrintF Format Specification (*continued*)**

Character	Description
*	Indicates that the next argument (must be an integer) in the list specifies the field width. In this case, the argument following that one is used for the value of this field. Example: <code>StrPrintF(s, "%*d%d", 4, 8, 5);</code> Output to s: 8 5
<number>	Specifies a minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces on the left (or right, if the left justified flag is also specified) to fill out the field width. Example: <code>StrPrintF(s, "%d%5d", 4, 3);</code> Output to s: 4 3
h	Specifies that the following d, i, u, or x conversion corresponds to a short or unsigned short argument. Example: <code>StrPrintF(s, "%hd", 401);</code> Output to s: 401
l or L	Specifies that the following d, i, u, x, or c conversion corresponds to a long or unsigned long <code>StrPrintF(s, "%ld", 999999999);</code> Output to s: 999999999
<character>	A character that indicates the type of conversion to be performed. The supported conversion characters include:

**Table 47.1 StrVPrintF Format Specification (*continued*)**

<b>Character</b>	<b>Description</b>
d or i	A signed integer argument is converted to decimal notation. Example: <code>StrPrintF(s, "%d %d", 4, -4);</code> Output to s: 4 -4
u	An unsigned integer argument is converted to decimal notation. Example: <code>StrPrintF(s, "%u %u", 4, -4);</code> Output to s: 4 65532
x	An integer argument is converted to hexadecimal notation. Example: <code>StrPrintF(s, "%x", 125);</code> Output to s: 0000007D
s	A string (char *) argument is copied to the destination string. Example: <code>StrPrintF(s, "ABC%s", "DEF");</code> Output to s: ABCDEF
c or C	A single character argument is copied to the destination string. If C is used or if the l modifier is used, the argument must be a WChar. Example: <code>StrPrintF(s, "Telephone%c", 's');</code> Output to s: Telephones
%	A % character is copied to the destination string. Example: <code>StrPrintF(s, "%%");</code> Output to s: %

**Example** Here's an example of how to use this call:

## String Manager

### *String Manager Functions*

---

```
#include <unix_stdarg.h>
void MyPrintF(Char *s, Char *formatStr, ...)
{
    va_list args;
    Char text[0x100];
    va_start(args, formatStr);
    StrVPrintF(text, formatStr, args);
    va_end(args);
    MyPutS(text);
}
```

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

**See Also** [StrPrintF](#)

# System Event Manager

---

This chapter describes functions available in the system event manager. The system event manager API is declared in the header files `Event.h` and `SysEvtMgr.h`.

For more information on the system event manager, see the chapter “[Event Loop](#)” in the *Palm OS Programmer’s Companion*, vol. I. The reference for specific events sent by the system are documented in “[Palm OS Events](#).”

## System Event Manager Data Structures

The following system event manager data structures are documented in the “Palm OS Events” chapter:

- [eventsEnum](#)
- [EventType](#)
- [EventPtr](#)

# System Event Manager Functions

## EvtAddEventToQueue

**Purpose** Add an event to the event queue.

**Declared In** Event.h

**Prototype** void EvtAddEventToQueue (const EventType \*event)

**Parameters** -> event Pointer to the structure that contains the event.

**Result** Returns nothing.

**Comments** This function makes a copy of the structure that you pass in and adds it to the event queue.

## EvtAddUniqueEventToQueue

**Purpose** Add an event to the event queue, replacing one of the same type if it is found.

**Declared In** Event.h

**Prototype** void EvtAddUniqueEventToQueue  
(const EventType \*eventP, UInt32 id,  
Boolean inPlace)

**Parameters** -> eventP Pointer to the structure that contains the event.

-> id ID of the event. 0 means match only on the type.

-> inPlace If true, any existing event is replaced. If false, the existing event is deleted and a new event is added to end of queue.

**Result** Returns nothing.

**Comments** This function looks for an event in the event queue of the same event type and ID (if specified). The routine replaces it with the new event, if found.

If no existing event is found, the new event is copied to the queue.

If an existing event is found, the routine proceeds as follows:

- If `inPlace` is `true`, the existing event is replaced with a copy of the new event.
- If `inPlace` is `false`, the existing event is removed and the new event is added to the end of the queue.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## EvtCopyEvent

**Purpose** Copy an event.

**Declared In** Event.h

**Prototype** void EvtCopyEvent (const EventType \*source,  
EventType \*dest)

**Parameters** -> source Pointer to the structure containing the event to copy.  
<- dest Pointer to the structure to copy the event to.

**Result** Returns nothing.

**Comments** Use this function only if you want to create an event that has the same type as the source event. The data field in an [EventType](#) structure is specific to events of a given type. If you were to use this function to copy a [keyDownEvent](#) and then change it to a [frmLoadEvent](#), the resulting frmLoadEvent would not have the proper data field.

If you want to create an event of a different type, do not use EvtCopyEvent. First clear the EventType structure using [MemSet](#) and then change the event type:

```
MemSet(&event, sizeof(EventType), 0);  
event.eType = frmLoadEvent;  
event.data.frmLoad.formID = formID;  
EvtAddEventToQueue(&event);
```

## EvtDequeuePenPoint

**Purpose** Get the next pen point out of the pen queue. This function is called by recognizers.

**Declared In** SysEvtMgr.h

**Prototype** Err EvtDequeuePenPoint (PointType \*retP)

**Parameters** <- retP      Return point.

**Result** Always returns 0.

**Comments** Called by a recognizer that wishes to extract the points of a stroke. Returns the point (-1, -1) at the end of a stroke.

Before calling this routine, you must call  
[EvtDequeuePenStrokeInfo](#).

## EvtDequeuePenStrokeInfo

**Purpose** Initiate the extraction of a stroke from the pen queue.

**Declared In** SysEvtMgr.h

**Prototype** Err EvtDequeuePenStrokeInfo (PointType \*startPtP,  
PointType \*endPtP)

**Parameters** <- startPtP      Start point returned here.  
<- endPtP      End point returned here.

**Result** Always returns 0.

**Comments** Called by the system function EvtGetSysEvent when a [penUpEvent](#) is being generated. This routine must be called before [EvtDequeuePenPoint](#) is called.

Subsequent calls to EvtDequeuePenPoint return points at the starting point in the stroke and including the end point. After the

end point is returned, the next call to [EvtDequeuePenPoint](#) returns the point -1, -1.

**See Also** [EvtDequeuePenPoint](#)

## EvtEnableGraffiti

**Purpose** Set Graffiti® enabled or disabled.

**Declared In** SysEvtMgr.h

**Prototype** void EvtEnableGraffiti (Boolean enable)

**Parameters** -> enable true to enable Graffiti, false to disable Graffiti.

**Result** Returns nothing.

## EvtEnqueueKey

**Purpose** Place keys into the key queue.

**Declared In** SysEvtMgr.h

**Prototype** Err EvtEnqueueKey (WChar ascii, UInt16 keycode, UInt16 modifiers)

**Parameters** -> ascii Character code for the key.  
-> keycode Virtual key code of key. This is the keyCode field of the [keyDownEvent](#) and is currently unused.  
-> modifiers Modifiers for keyDownEvent.

**Result** Returns 0 if successful, or evtErrParamErr if an error occurs.

<b>Comments</b>	This function disables interrupts while the queue header is being modified because both interrupt- and non-interrupt-level code can post keys into the queue.  Entries in the key queue only take 1 byte if the <code>ascii</code> parameter has a value less than 256 and the <code>keycode</code> and <code>modifiers</code> parameters are both zero. Otherwise an entry can take up to 7 bytes.
-----------------	---

---

**IMPORTANT:** Make sure you pass a `WChar` as the `ascii` parameter, not a `Char`. If you pass a high-ASCII `Char`, the compiler sign-extends it to be a 16-bit value, resulting in the wrong character being added to the key queue.

---

## EvtEventAvail

<b>Purpose</b>	Return <code>true</code> if an event is available.
<b>Declared In</b>	<code>Event.h</code>
<b>Prototype</b>	<code>Boolean EvtEventAvail (void)</code>
<b>Parameters</b>	None.
<b>Result</b>	Returns <code>true</code> if an event is available, <code>false</code> otherwise.
<b>Compatibility</b>	Implemented only if <a href="#">2.0 New Feature Set</a> is present.

## EvtFlushKeyQueue

**Purpose** Flush all keys out of the key queue.

**Declared In** SysEvtMgr.h

**Prototype** Err EvtFlushKeyQueue (void)

**Parameters** None.

**Result** Always returns 0.

**Comments** Called by the system function EvtSetPenQueuePtr.

## EvtFlushNextPenStroke

**Purpose** Flush the next stroke out of the pen queue.

**Declared In** SysEvtMgr.h

**Prototype** Err EvtFlushNextPenStroke ()

**Parameters** None.

**Result** Always returns 0.

**Comments** Called by recognizers that need only the start and end points of a stroke. If a stroke has already been partially dequeued (by [EvtDequeuePenStrokeInfo](#)) this routine finishes the stroke dequeuing. Otherwise, this routine flushes the next stroke in the queue.

**See Also** [EvtDequeuePenPoint](#)

## EvtFlushPenQueue

**Purpose** Flush all points out of the pen queue.

**Declared In** SysEvtMgr.h

**Prototype** Err EvtFlushPenQueue (void)

**Parameters** None

**Result** Always returns 0.

**Comments** Called by the system function EvtSetKeyQueuePtr.

**See Also** [EvtPenQueueSize](#)

## EvtGetEvent

**Purpose** Return the next available event.

**Declared In** Event.h

**Prototype** void EvtGetEvent (EventType \*event, Int32 timeout)

**Parameters** <- event Pointer to the structure to hold the event returned.

-> timeout Maximum number of ticks to wait before an event is returned (evtWaitForever means wait indefinitely).

**Comments** Pass evtWaitForever as the timeout in most instances. When running on the device, this makes the CPU go into doze mode until the user provides input. For applications that do animation, pass a timeout value greater than or equal to zero.

Note that a timeout value greater than or equal to zero is simply the *maximum* number of ticks which can elapse before EvtGetEvent returns an event. If any other event—including a nilEvent—

occurs before this time has elapsed, EvtGetEvent will return that event. Otherwise, once the specified time has elapsed EvtGetEvent generates and returns a nilEvent. If you supply a value of zero for the timeout parameter, EvtGetEvent returns the event currently in the queue, or, if there aren't any events in the queue, it immediately generates and returns a nilEvent.

**Result** Returns nothing.

## EvtGetPen

**Purpose** Return the current status of the pen.

**Declared In** Event.h

**Prototype** void EvtGetPen (Int16 \*pScreenX, Int16 \*pScreenY, Boolean \*pPenDown)

**Parameters** <- pScreenX      x location relative to display.  
                <- pScreenY      y location relative to display.  
                <- pPenDown      true or false.

**Result** Returns nothing.

**Comments** Called by various UI routines.

**See Also** [EvtGetPenNative](#), [Key.currentState](#)

## EvtGetPenBtnList

**Purpose** Return a pointer to the silk-screen button array.

**Declared In** SysEvtMgr.h

**Prototype** const PenBtnInfoType \*EvtGetPenBtnList  
(UInt16 \*numButtons)

**Parameters** <- numButtons The number of elements in the returned array.

**Result** Returns a pointer to an array of the silk-screen buttons.

**Comments** This function returns an array of PenBtnInfoType structures:

```
typedef struct PenBtnInfoType {  
    RectangleType boundsR;  
    WChar asciiCode;  
    UInt16 keyCode;  
    UInt16 modifiers;  
} PenBtnInfoType;
```

The fields in the PenBtnInfoType contain the following information:

boundsR The button's bounding rectangle.

asciiCode The character code generated when the button is tapped. This is typically a virtual character.

keyCode Currently unused.

modifiers Modifiers for the key down event. (See the description of the modifiers field for [keyDownEvent](#).)

The number of buttons is device-dependent. Even devices with the same Palm OS® version may have differing numbers of silk-screen buttons. For example, Japanese devices typically have four extra

silk-screen buttons used to transliterate characters into different alphabets.

**See Also** [EvtProcessSoftKeyStroke](#)



### EvtGetPenNative

**Purpose** Get the current status of the pen using a window's active coordinate system.

**Declared In** Window.h

**Prototype**

```
void EvtGetPenNative (WinHandle winH,  
                      Int16 *pScreenX, Int16 *pScreenY,  
                      Boolean *pPenDown)
```

**Parameters**

-> winH	Handle to a valid window.
<- pScreenX	x location relative to the window.
<- pScreenY	y location relative to the window.
<- pPenDown	true if the pen is down, false otherwise.

**Result** Returns nothing.

**Comments** This function is a variation on [EvtGetPen](#). EvtGetPen returns a pen sample using the standard coordinate system, relative to the draw window, whereas EvtGetPenNative returns a pen sample using the active coordinate system of winH, relative to the window origin. If the active coordinate system is high density, the returned pen sample uses high-density coordinates.

On a debug ROM this function displays an error if winH doesn't reference a valid window object.

**Compatibility** Implemented only if the [High-Density Display Feature Set](#) is present.

## EvtGetSilkscreenAreaList

**Purpose** Returns a pointer to the silk-screen area array. This array contains the bounds of each silk-screen area.

**Declared In** SysEvtMgr.h

**Prototype** const SilkscreenAreaType  
\*EvtGetSilkscreenAreaList (UInt16 \*numAreas)

**Parameters** <- numAreas The number of elements in the returned array.

**Result** Returns a pointer to an array containing the bounds of each silk-screen area.

**Comments** This function returns an array of the SilkscreenAreaType structures:

```
typedef struct SilkscreenAreaType {  
    RectangleType bounds;  
    UInt32     areaType;  
    UInt16     index;  
} SilkscreenAreaType;
```

The fields in this structure provide the following information.

bounds The area's bounds.

areaType    The area type, can be one of the following:  
                  silkscreenRectGraffiti  
                            The Graffiti area.  
                  silkscreenRectScreen  
                            The entire silkscreen area.  
Depending on the handheld manufacturer, Palm Powered™ devices may have other area types.

index        If the area type is silkscreenRectGraffiti, this field is either alphaGraffitiIsSilkscreenArea to represent the portion where letters are entered or numericGraffitiIsSilkscreenArea to represent the portion where numbers are entered.

**Compatibility**    Implemented only if [3.5 New Feature Set](#) is present. If [5.0 New Feature Set](#) is present, this function should be considered “System Use Only”; applications should do what they can to avoid using it.

## EvtKeydownIsVirtual

**Purpose**    Macro that indicates if eventP is a pointer to a virtual character key down event.

**Declared In**    Event.h

**Prototype**    EvtKeydownIsVirtual (eventP)

**Parameters**    -> eventP        Pointer to an [EventType](#) structure.

**Result**    Returns true if the character is a letter in an alphabet or a numeric digit, false otherwise.

**Comments**    The macro assumes that the caller has already determined the event is a keyDownEvent.

This macro is intended for use by the system. Applications should use [TxtGlueCharIsVirtual](#), contained in the [PalmOSGlue Library](#).

**Compatibility** Implemented in the Palm OS 3.5 SDK.

**See Also** [TxtGlueCharIsVirtual](#)

## EvtKeyQueueEmpty

**Purpose** Return true if the key queue is currently empty.

**Declared In** SysEvtMgr.h

**Prototype** Boolean EvtKeyQueueEmpty (void)

**Parameters** None.

**Result** Returns true if the key queue is currently empty, otherwise returns false.

**Comments** Usually called by the key manager to determine if it should enqueue auto-repeat keys.

## EvtKeyQueueSize

**Purpose** Return the size of the current key queue in bytes.

**Declared In** SysEvtMgr.h

**Prototype** UInt32 EvtKeyQueueSize (void)

**Parameters** None.

**Result** Returns size of queue in bytes.

**Comments** Called by applications that wish to see how large the current key queue is.

## EvtPenQueueSize

- Purpose** Return the size of the current pen queue in bytes.
- Declared In** SysEvtMgr.h
- Prototype** `UIInt32 EvtPenQueueSize (void)`
- Parameters** None.
- Result** Returns size of queue in bytes.
- Comments** Call this function to see how large the current pen queue is.

## EvtProcessSoftKeyStroke

- Purpose** Translate a stroke in the system area of the digitizer and enqueue the appropriate key events in to the key queue.
- Declared In** SysEvtMgr.h
- Prototype** `Err EvtProcessSoftKeyStroke (PointType *startPtP,  
PointType *endPtP)`
- Parameters** `-> startPtP` Start point of stroke.  
`-> endPtP` End point of stroke.
- Result** Returns 0 if recognized, -1 if not recognized.
- See Also** [EvtGetPenBtnList](#), [GrfProcessStroke](#)

## EvtResetAutoOffTimer

<b>Purpose</b>	Reset the auto-off timer.
<b>Declared In</b>	SysEvtMgr.h
<b>Prototype</b>	Err EvtResetAutoOffTimer (void)
<b>Parameters</b>	None.
<b>Result</b>	Always returns 0.
<b>Comments</b>	<p>Called by the serial link manager; can be called periodically by other managers.</p> <p>EvtResetAutoOffTimer resets the auto-off timer so that the device does not turn off until at least the default amount of idle time has passed. You can use this function to ensure that the device doesn't automatically power off during a long operation without user input (for example, when there is a lot of serial port activity).</p> <p>If you need more control over the auto-off timer and the <a href="#">3.5 New Feature Set</a> is present, consider using <a href="#">EvtSetAutoOffTimer</a> instead of this function.</p>
<b>See Also</b>	<a href="#">SysSetAutoOffTime</a>

## EvtSetAutoOffTimer

<b>Purpose</b>	Set the auto-off timer.
<b>Declared In</b>	SysEvtMgr.h
<b>Prototype</b>	Err EvtSetAutoOffTimer (EvtSetAutoOffCmd cmd, UInt16 timeout)
<b>Parameters</b>	-> cmd One of the defined commands.

## System Event Manager

### *System Event Manager Functions*

---

-> timeout      A new timeout value in seconds. If cmd is ResetTimer, this parameter is ignored.

**Result**      Always returns errNone.

**Comments**      Use EvtSetAutoOffTimer to ensure that the device doesn't automatically power off during a long operation that has no user input (for example, when there is a lot of serial port activity).

The cmd parameter specifies the operation you want to perform. It takes one of the following EvtSetAutoOffCmd constants:

SetAtLeast      Make sure that the device won't turn off until timeout seconds of idle time has passed.  
(This operation only changes the current value if it's less than the value you specify.)

SetExactly      Set the timer to turn off in timeout seconds

SetAtMost      Make sure the device will turn before timeout seconds has passed. (This operation only changes the current value if it's greater than the value you specify.)

SetDefault      Change the default auto-off timeout to timeout seconds.

ResetTimer      Reset the auto-off timer so that the device does not turn off until at least the default seconds of idle time has passed.

**Compatibility**      Implemented only if [3.5 New Feature Set](#) is present.

**See Also**      [EvtResetAutoOffTimer](#), [SysSetAutoOffTime](#)

## EvtSetNullEventTick

<b>Purpose</b>	Make sure a <a href="#">nilEvent</a> occurs in at least the specified number of ticks.	
<b>Declared In</b>	SysEvtMgr.h	
<b>Prototype</b>	Boolean EvtSetNullEventTick (UInt32 tick)	
<b>Parameters</b>	<b>-&gt; tick</b> Maximum number of system ticks that should elapse before a nilEvent is added to the queue.	
<b>Result</b>	Returns <code>true</code> if timeout value changed, or <code>false</code> if it did not change.	
<b>Compatibility</b>	In versions prior to Palm OS 3.5, this function was implemented as a macro.	

## EvtSysEventAvail

<b>Purpose</b>	Return <code>true</code> if a low-level system event (such as a pen or key event) is available.	
<b>Declared In</b>	SysEvtMgr.h	
<b>Prototype</b>	Boolean EvtSysEventAvail (Boolean ignorePenUps)	
<b>Parameters</b>	<b>ignorePenUps</b> If <code>true</code> , this routine ignores pen-up events when determining if there are any system events available.	
<b>Result</b>	Returns <code>true</code> if a system event is available.	
<b>Comment</b>	Call <a href="#">EvtEventAvail</a> to determine whether high-level software events are available.	
<b>Compatibility</b>	Implemented only if <a href="#">2.0 New Feature Set</a> is present.	

## EvtWakeup

<b>Purpose</b>	Force the event manager to wake up and send a <a href="#">nilEvent</a> to the current application.
<b>Declared In</b>	SysEvtMgr.h
<b>Prototype</b>	Err EvtWakeup (void)
<b>Parameters</b>	None.
<b>Result</b>	Always returns 0.
<b>Comments</b>	Called by interrupt routines, like the sound manager and alarm manager.
<b>See Also</b>	<a href="#">EvtWakeupWithoutNilEvent</a>

## EvtWakeupWithoutNilEvent

<b>Purpose</b>	Force the event manager to wake up without sending a <a href="#">nilEvent</a> to the current application.
<b>Declared In</b>	SysEvtMgr.h
<b>Prototype</b>	Err EvtWakeupWithoutNilEvent ()
<b>Parameters</b>	None.
<b>Result</b>	Always returns 0.
<b>Comments</b>	Called by interrupt routines.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">EvtWakeup</a>

# System Manager

---

This chapter provides reference material for the system manager. The system manager API is declared in the header files `SystemMgr.h` and `SysUtils.h`.

For more information on the system manager, see the chapters “[Application Startup and Stop](#)” and “[Palm System Support](#)” in the *Palm OS Programmer’s Companion*, vol. I.

## System Manager Data Structures

### **SysDBListItemType**

The `SysDBListItemType` structure describes a single database or panel. The [`SysCreateDataBaseList`](#) and [`SysCreatePanelList`](#) functions each create and return an array of `SysDBListItemType` structures.

```
typedef struct {
    Char          name [dmDBNameLength] ;
    UInt32        creator;
    UInt32        type;
    UInt16        version;
    LocalID       dbID;
    UInt16        cardNo;
    BitmapPtr     iconP;
} SysDBListItemType;
```

## System Manager

### *System Functions*

---

# System Functions

## SysAppLaunch

**Purpose** Launch a specified application as a subroutine of the caller.

**Declared In** SystemMgr.h

**Prototype** Err SysAppLaunch (UInt16 cardNo, LocalID dbID,  
UInt16 launchFlags, UInt16 cmd, MemPtr cmdPBP,  
UInt32 \*resultP)

**Parameters**

-> cardNo, dbID	The card number and ID of the resource database of the application to launch.
-> launchFlags	Set to 0.
-> cmd	Launch code.
-> cmdPBP	Launch code parameter block.
<- resultP	The value returned from the application's <a href="#">PilotMain</a> routine.

**Result** Returns 0 if no error, or one of `sysErrParamErr`, `memErrNotEnoughSpace`, or `sysErrOutOfOwnerIDs`.

**Comments** Applications can use SysAppLaunch to send a specific launch code to another application and have control return to the calling application when finished. This function in effect makes the specified application a subroutine of the caller. If you want to actually close your application and call another application, use [SysUIAppSwitch](#) instead of this function. SysUIAppSwitch sends the current application an `appStopEvent` and then starts the specified application.

Do not use this function to open the system-supplied Application Launcher application. If another application has replaced the default launcher with one of its own, this function will open the custom launcher instead of the system-supplied one. To open the system-supplied launcher reliably, enqueue a `keyDownEvent` that

contains a launchChr, as shown in the section “[Application Launcher](#)” of the user interface chapter in the *Palm OS Programmer’s Companion*, vol. I.

---

**NOTE:** For important information regarding the correct use of this function, see the “[Application Startup and Stop](#)” chapter in the *Palm OS Programmer’s Companion*, vol. I.

---

**See Also**

[SysBroadcastActionCode](#), [SysUIAppSwitch](#),  
[SysCurAppDatabase](#)

## SysBatteryInfo

**Purpose** Retrieve settings for the batteries. Set set to `false` to retrieve battery settings. (Applications should **not** change any of the settings).

---

**WARNING!** Use this function only to **retrieve** settings!

---

**Declared In** SystemMgr.h

**Prototype**

```
UInt16 SysBatteryInfo (Boolean set,  
                      UInt16 *warnThresholdP,  
                      UInt16 *criticalThresholdP, Int16 *maxTicksP,  
                      SysBatteryKind *kindP, Boolean *pluggedIn,  
                      UInt8 *percentP)
```

**Parameters**

set	If <code>false</code> , parameters with non-NULL pointers are retrieved. Never set this parameter to <code>true</code> .
warnThresholdP	Pointer to battery voltage warning threshold in volts*100, or NULL.
criticalThresholdP	Pointer to the battery voltage critical threshold in volts*100, or NULL.
maxTicksP	Pointer to the battery timeout, or NULL.

## System Manager

### *System Functions*

---

kindP	Pointer to the battery kind, or NULL.
pluggedIn	Pointer to pluggedIn return value, or NULL.
percentP	Percentage of power remaining in the battery.

**Result** Returns the current battery voltage in volts\*100.

**Comments** Call this function to make sure an upcoming activity won't be interrupted by a low battery warning.  
warnThresholdP and maxTicksP are the battery-warning voltage threshold and time out. If the battery voltage falls below the threshold, or the timeout expires, a lowBatteryChr key event is put on the queue. Normally, applications call [SysHandleEvent](#) which calls SysBatteryDialog in response to this event.  
criticalThresholdP is the battery voltage threshold. If battery voltage falls below this level, the system turns itself off without warning and doesn't turn on until battery voltage is above it again.

**Compatibility** This function was revised for Palm OS® 3.0. In Palm OS 3.0, the percentP parameter was added. This enhancement is implemented only if [3.0 New Feature Set](#) is present.

**See Also** [SysBatteryInfoV20](#)

## SysBatteryInfoV20

**Purpose** Retrieve settings for the batteries. Set to `false` to retrieve battery settings. (Applications should **not** change any of the settings).

---

**WARNING!** Use this function only to **retrieve** settings!

---

**Declared In** SystemMgr.h

**Prototype** UInt16 SysBatteryInfoV20 (Boolean set,  
                  UInt16 \*warnThresholdP,  
                  UInt16 \*criticalThresholdP, Int16 \*maxTicksP,  
                  SysBatteryKind \*kindP, Boolean \*pluggedIn)

**Parameters** `set` If false, parameters with non-NULL pointers are retrieved. Never set this parameter to true.

`warnThresholdP` Pointer to battery voltage warning threshold in volts\*100, or NULL.

`criticalThresholdP` Pointer to the battery voltage critical threshold in volts\*100, or NULL.

`maxTicksP` Pointer to the battery timeout, or NULL.

`kindP` Pointer to the battery kind, or NULL.

`pluggedIn` Pointer to `pluggedIn` return value, or NULL.

**Result** Returns the current battery voltage in volts\*100.

**Comments** Call this function to make sure an upcoming activity won't be interrupted by a low battery warning.

`warnThresholdP` and `maxTicksP` are the battery-warning voltage threshold and time out. If the battery voltage falls below the threshold, or the timeout expires, a `lowBatteryChr` key event is put on the queue. Normally, applications call [SysHandleEvent](#) which calls `SysBatteryDialog` in response to this event.

`criticalThresholdP` is the battery voltage threshold. If battery voltage falls below this level, the system turns itself off without warning and doesn't turn on until battery voltage is above it again.

**Compatibility** This function corresponds to the Palm OS 2.0 version of `SysBatteryInfo`. Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [SysBatteryInfo](#)

## SysBinarySearch

**Purpose** Search elements in a sorted array for the specified data according to the specified comparison function.

**Declared In** SysUtils.h

**Prototype** Boolean SysBinarySearch (void const \*baseP,  
Int16 numElements, Int16 width,  
SearchFuncPtr searchF, void const \*searchData,  
Int32 other, Int32 \*position, Boolean findFirst)

<b>Parameters</b>	baseP	Base pointer to an array of elements
	numOfElements	Number of elements to search. Must be greater than 0.
	width	Width of each array element.
	searchF	Search function.
	searchData	Data to search for. This data is passed to the searchF function.
	other	Data to be passed as the third parameter (the other parameter) to the comparison function.
	position	Pointer to the position result.
	findFirst	If set to true, the first matching element is returned. Use this parameter if the array contains duplicate entries to ensure that the first such entry will be the one returned.
<b>Result</b>	Returns true if an exact match was found. In this case, position points to the element number where the data was found. Returns false if an exact match was not found. If false is returned, position points to the element number where the data should be inserted if it was to be added to the array in sorted order.	
<b>Comments</b>	<p>The array must be sorted in ascending order prior to the search. Use <a href="#">SysInsertionSort</a> or <a href="#">SysQSort</a> to sort the array.</p> <p>The search starts at element 0 and ends at element (numOfElements - 1).</p> <p>The search function's (searchF) prototype is:</p> <pre>Int16 _searchF (void const *searchData,                  void const *arrayData, Int32 other);</pre> <p>The first parameter is the data for which to search, the second parameter is a pointer to an element in the array, and the third parameter is any other necessary data.</p>	

## System Manager

### *System Functions*

---

The function returns:

- > 0 if the search data is greater than the element
- < 0 if the search data is less than the element
- 0 if the search data is the same as the element

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## SysBroadcastActionCode

**Purpose** Send the specified action code (launch code) and parameter block to the latest version of every UI application.

**Declared In** SystemMgr.h

**Prototype** Err SysBroadcastActionCode (UIInt16 cmd,  
MemPtr cmdPBP)

**Parameters** cmd Action code to send.

cmdPBP Action code parameter block to send.

**Result** Returns 0 if no error, or one of the following errors:  
sysErrParamErr, memErrNotEnoughSpace, or  
sysErrOutOfOwnerIDs.

**See Also** [SysAppLaunch](#), [Chapter 2, “Application Startup and Stop.”](#) of the *Palm OS Programmer’s Companion*, vol. I

## SysCopyStringResource

**Purpose** Copy a resource string to a passed string.

**Declared In** SysUtils.h

**Prototype** void SysCopyStringResource (Char \*string,  
Int16 theID)

**Parameters** string String to copy the resource string to.  
theID Resource string ID.

**Result** Stores a copy of the resource string in `string`.

## SysCreateDataBaseList

**Purpose** Generate a list of databases found on the memory cards matching a specific type and return the result. If `lookupName` is `true` then a name in a `tAIN` resource is used instead of the database's name and the list is sorted. Only the last version of a database is returned. Databases with multiple versions are listed only once.

**Declared In** `SystemMgr.h`

**Prototype** `Boolean SysCreateDataBaseList (UInt32 type,  
                  UInt32 creator, UInt16 *dbCount, MemHandle *dbIDs,  
                  Boolean lookupName)`

**Parameters**

<code>type</code>	The type of database to find. Use 0 to find all databases.
<code>creator</code>	The creator ID of the database to find. Use 0 to find any creator ID.
<code>dbCount</code>	A pointer to an integer value that is updated by this function to the number of matching databases.
<code>dbIDs</code>	A pointer to a handle that gets allocated to contain the database list. Upon return, this references an array of <a href="#"><code>SysDBListItemType</code></a> structures. See the Comments section below for more information.
<code>lookupName</code>	If <code>true</code> , <code>SysCreateDatabaseList</code> uses <code>tAIN</code> names and sorts the list.

**Result** Returns `false` if no databases were found, and `true` if any databases were found. The value of `dbCount` is updated to reflect the number of databases that were found. If at least one database is found, `dbIDs` is updated to reference a array of [`SysDBListItemType`](#) structures; this array contains `dbCount` items.

**Comments**

This function creates a list of unique databases, where unique is defined as having a different type and creator ID. Two or more databases with the same type and creator ID are counted as one. Thus, you cannot use `SysCreateDataBaseList` to build a list of databases that share a common type and creator. There are two exceptions to this rule, however. If type is 0 or if creator is not 0, the code that removes “non-unique” databases isn’t run. It also isn’t run if the type is `sysFileTpqa`, since web-clipping databases all have the same type and creator ID.

If this function returns `true` and the value of `dbCount` is greater than 0, than you can iterate through the list of database items, as shown in [Listing 49.1](#)

**Listing 49.1 Using the `SysCreateDatabaseList` function**

---

```
SysDBListItemType          *dbListIDsP;
MemHandle                  dbListIDsH;
UInt16                     dbCount = 0;
Boolean                    status;
UInt16                     counter;
SysDBListItemType          theItem;

status = SysCreateDatabaseList(sysFileTpqa, 0,
                               &dbCount, &dbListIDsH, false);

if (status == true && dbCount > 0)
{
    dbListIDsP = MemHandleLock (dbListIDsH);
    for (counter = 0; counter < dbCount; counter++)
        if StrCompare(dbListIDsP[counter].name,
                       "MINE") == 0
            // we found my database
    ...
    ...
    MemPtrFree (dbListIDsP);
}
```

---

**NOTE:** It is your responsibility to free the memory allocated by this function for the list of databases.

---

**Compatibility**

Implemented only if [2.0 New Feature Set](#) is present.

## SysCreatePanelList

<b>Purpose</b>	Generate a list of panels found on the memory cards and return the result. Multiple versions of a panel are listed once.
<b>Declared In</b>	SystemMgr.h
<b>Prototype</b>	Boolean SysCreatePanelList (UInt16 *panelCount, MemHandle *panelIDs)
<b>Parameters</b>	panelCount      Pointer to set to the number of panels. panelIDs      A pointer to a handle that gets allocated to contain the panel list. Upon return, this references an array of <a href="#">SysDBListItemType</a> structures.
<b>Result</b>	Returns <code>false</code> if no panels were found, and <code>true</code> if any panels were found. The value of <code>panelCount</code> is updated to reflect the number of panels that were found. If at least one panel is found, <code>panelIDs</code> is updated to reference a array of <a href="#">SysDBListItemType</a> structures; this array contains <code>panelCount</code> items.
<b>Comments</b>	If this function returns <code>true</code> and the value of <code>panelCount</code> is greater than 0, than you can iterate through the list of panel items, as shown in <a href="#">Listing 49.1</a> . It is your responsibility to free the memory allocated for the panel list.
<b>Compatibility</b>	Implemented only if <a href="#">2.0 New Feature Set</a> is present.

## SysCurAppDatabase

**Purpose** Return the card number and database ID of the current application's resource database.

**Declared In** SystemMgr.h

**Prototype** Err SysCurAppDatabase (UInt16 \*cardNoP,  
LocalID \*dbIDP)

**Parameters** cardNoP Pointer to the card number; 0 or 1.  
dbIDP Pointer to the database ID.

**Result** Returns 0 if no error, or `SysErrParamErr` if an error occurs.

**See Also** [SysAppLaunch](#), [SysUIAppSwitch](#)

## SysErrString

**Purpose** Returns text to describe an error number. This routine looks up the textual description of a system error number in the appropriate List resource and creates a string that can be used to display that error.

The actual string will be of the form: "<error message> (XXXX)" where XXXX is the hexadecimal error number.

This routine looks for a resource of type 'tstl' and resource ID of (err>>8). It then grabs the string at index (err & 0x00FF) out of that resource.

The first string in the resource is called index #1 by Constructor, NOT #0. For example, an error code of 0x0101 will fetch the first string in the resource.

**Declared In** SysUtils.h

**Prototype** Char \*SysErrString (Err err, Char \*strP,  
UInt16 maxLen)

**Parameters** err Error number

`strP`              Pointer to space to form the string  
`maxLength`      Size of `strP` buffer.

**Result** Stores the error number string.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

# SysFormPointerArrayToStrings

**Purpose** Form an array of pointers to strings in a block. Useful for setting the items of a list.

**Declared In** SysUtils.h

**Prototype** MemHandle SysFormPointerArrayToStrings (Char \*c, Int16 stringCount)

<b>Parameters</b>	c	Pointer to packed block of strings, each terminated by a null character.
	stringCount	Count of strings in block.

**Result** Unlocked handle to allocated array of pointers to the strings in the passed block. The returned array points to the strings in the passed packed block. Note that you'll need to free the returned handle when you no longer need it.

#### **See Also**    [LstSetListChoices](#)



## New **sysFtrNumProcessorIs68K**

**Purpose** Macro that determines whether or not the underlying processor is part of the 68K family.

**Declared In** SystemMgr.h

**Prototype**

```
#define sysFtrNumProcessorIs68K(x)
((x&sysFtrNumProcessor68KIfZero)==0)? true :
false)
```

**Parameters** -> x Processor type obtained from a call to [FtrGet](#).

**Result** Returns true if the underlying processor is a 68K, false otherwise.

**Comments** This macro is typically used in conjunction with [PceNativeCall](#).

**Example**

```
UInt32 processorType;

FtrGet(sysFileCSystem, sysFtrNumProcessorID, &processorType);
if (sysFtrNumProcessorIs68K(processorType)) {
    // processor is 68K
} else {
    // processor is not 68K
}
```

---



**New**

## sysFtrNumProcessorIsARM

**Purpose** Macro that determines whether or not the underlying processor is part of the ARM family.

**Declared In** SystemMgr.h

**Prototype**

```
#define sysFtrNumProcessorIsARM(x)
((x&sysFtrNumProcessorARMIfNotZero)!=0) ? true :
false)
```

**Parameters** -> x Processor type obtained from a call to [FtrGet](#).

**Result** Returns `true` if the underlying processor is an ARM core, `false` otherwise.

**Comments** This macro is typically used in conjunction with [PceNativeCall](#).

**Example**

```
UInt32 processorType;

FtrGet(sysFileCSystem, sysFtrNumProcessorID, &processorType);
if (sysFtrNumProcessorIsARM(processorType)) {
    // processor is ARM
} else {
    // processor is not ARM
}
```

---

## SysGetOSVersionString

<b>Purpose</b>	Return the version number of the Palm™ operating system.
<b>Declared In</b>	SystemMgr.h
<b>Prototype</b>	Char *SysGetOSVersionString()
<b>Parameters</b>	None.
<b>Result</b>	Returns a string such as "v. 3.0."
<b>Comments</b>	You must free the returned string using the <a href="#">MemPtrFree</a> function.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## SysGetROMToken

<b>Purpose</b>	Return from ROM a value specified by token.	
<b>Declared In</b>	SystemMgr.h	
<b>Prototype</b>	Err SysGetROMToken (UInt16 cardNo, UInt32 token, UInt8 **dataP, UInt16 *sizeP)	
<b>Parameters</b>	-> cardNo	The card on which the ROM to be queried resides. Currently, no Palm hardware provides multiple cards, so this value must be 0.
	-> token	The value to retrieve, as specified by one of the following tokens:  <code>sysROMTokenSnum</code> The serial number of the ROM, expressed as a text string with no null terminator.
	<- dataP	Pointer to a text buffer that holds the requested value when the function returns.

## System Manager

### System Functions

---

<- sizeP                    The number of bytes in the dataP buffer.

**Result**    Returns the requested value if no error, or an error code if an error occurs. If this function returns an error, or if the returned pointer to the buffer is NULL, or if the first byte of the text buffer is 0xFF, then no serial number is available.

**Comments**    The serial number is shown to the user in the Application Launcher, along with a checksum digit you can use to validate input when your users read the ID from their device and type it in or tell it to someone else.

**Compatibility**    Implemented only if [3.0 New Feature Set](#) is present. Serial numbers are available only on flash ROM-based units.

**See Also**    “[Retrieving the ROM Serial Number](#)” section in the *Palm OS Programmer’s Companion*, vol. I shows how to retrieve the ROM serial number and calculate its associated checksum.

## SysGetStackInfo

**Purpose**    Return the start and end of the current thread’s stack.

**Declared In**    SystemMgr.h

**Prototype**    Boolean SysGetStackInfo (MemPtr \*startPP,  
                          MemPtr \*endPP)

**Parameters**    startPP                    Upon return, points to the start of the stack.  
                          endPP                    Upon return, points to the end of the stack.

**Result**    Returns `true` if the stack has not overflowed, that is, the value of the stack overflow address has not been changed. Returns `false` if the stack overflow value has been overwritten, meaning that a stack overflow has occurred.

**Compatibility**    Implemented only if [3.0 New Feature Set](#) is present.

## SysGetTrapAddress

<b>Purpose</b>	Return the address of a function given its system trap.
<b>Declared In</b>	SystemMgr.h
<b>Prototype</b>	<code>void *SysGetTrapAddress (UInt16 trapNum)</code>
<b>Parameters</b>	<code>-&gt; trapNum</code> One of the routine selectors defined in <code>SysTraps.h</code> ( <code>sysTrap...</code> ) or <code>CoreTraps.h</code> on Palm OS version 3.5 and higher.
<b>Result</b>	Returns the address of the corresponding function. Returns the address of the <code>sysTrapSysUnimplemented</code> routine if an invalid routine selector is passed; compare the results of this function to <code>SysGetTrapAddress (sysTrapSysUnimplemented)</code> if you need to check for an invalid routine selector.
<b>Comments</b>	<p>Use this function for performance reasons. You can then use the address it returns to call the function without having to go through the trap dispatch table. This function is mostly useful for optimizing the performance of functions called in a tight loop.</p> <p>The Palm OS trap dispatch mechanism allows the trap table entries to be modified at any time, either as the result of a system update or a hack. For this reason, it's important to call this function <b>immediately before</b> entering the tight loop. If the trap address changes in between when you call <code>SysGetTrapAddress</code> and you use the address, the wrong function will be called.</p>
<b>Compatibility</b>	On Palm OS 3.0 and earlier, this function contains a bug that causes it to return a garbage value if an invalid routine selector is passed. To prevent this bug from affecting your application, use <code>SysGlueGetTrapAddress</code> in the <code>PalmOSGlue</code> library instead of calling this function directly. For more information, see <a href="#">Chapter 75, “PalmOSGlue Library.”</a>

## SysGremlins

**Purpose** Query the Gremlins facility. You pass a selector for a function and parameters for that function. Gremlins performs the function call and returns the result.

**Declared In** SysUtils.h

**Prototype** `UInt32 SysGremlins (GremlinFunctionType selector,  
GremlinParamsType *params)`

**Parameters** `selector` The selector for a function to pass to Gremlins.  
`params` Pointer to a parameter block used to pass parameters to the function specified by `selector`.

**Result** Returns the result of the function performed in Gremlins.

**Comments** Currently, only one selector is defined, `GremlinIsOn`, which takes no parameters. `GremlinIsOn` returns 0 if Gremlins is not running, non-zero if it is running.

Currently, non-zero values are returned only from the version of Gremlins in the Palm OS emulator. The Gremlins running in the Simulator on a Macintosh and over the serial line via the Palm Debugger return zero for `GremlinIsOn`.

Use this function if you need to alter the application's behavior when Gremlins is running. For example, the debug 3.0 ROM refuses to bring up the digitizer panel when Gremlins is running under the emulator.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

In Palm OS 3.2 and later, `SysGremlins` is replaced by the functions defined in the file `HostControl.h`. Specifically, the one selector supported by `SysGremlins` is replaced with the function `HostGremlinIsRunning`. For backward compatibility, `SysGremlins` is mapped to `HostGremlinIsRunning`.

## SysHandleEvent

<b>Purpose</b>	Handle defaults for system events such as hard and soft key presses.
<b>Declared In</b>	SystemMgr.h
<b>Prototype</b>	Boolean SysHandleEvent (EventPtr eventP)
<b>Parameters</b>	eventP                    Pointer to an event.
<b>Result</b>	Returns true if the system handled the event.
<b>Comments</b>	Applications should call this routine immediately after calling <a href="#">EvtGetEvent</a> unless they want to override the default system behavior. However, overriding the default system behavior is almost never appropriate for an application.
<b>See Also</b>	<a href="#">EvtProcessSoftKeyStroke</a> , <a href="#">KeyRates</a>

## SysInsertionSort

<b>Purpose</b>	Sort elements in an array according to the passed comparison function.
<b>Declared In</b>	SysUtils.h
<b>Prototype</b>	void SysInsertionSort (void *baseP, Int16 numElements, Int16 width, CmpFuncPtr comparF, Int32 other)
<b>Parameters</b>	baseP                    Base pointer to an array of elements. numElements               Number of elements to sort (must be at least 2). width                     Width of an element. comparF                   Comparison function. See Comments, below.

## **System Manager**

### *System Functions*

---

other              Other data passed to the comparison function.

**Result**    Returns nothing.

**Comments** Only elements which are out of order move. Moved elements are moved to the end of the range of equal elements. If a large amount of elements are being sorted, try to use the quick sort (see [SysQSort](#)).

This is the insertion sort algorithm: Starting with the second element, each element is compared to the preceding element. Each element not greater than the last is inserted into sorted position within those already sorted. A binary search for the insertion point is performed. A moved element is inserted after any other equal elements.

In order to use `SysInsertionSort` you must write a comparison function with the following prototype:

```
Int16 comparF (void *p1, void *p2, Int32 other)
```

Your comparison function must return zero if p1 equals p2, a positive integer value if p1 is greater than p2, and a negative integer value if p1 is less than p2. Note that the value of the parameter named other is passed through from the `SysInsertionSort` call and can be used to control the behavior of the `comparF` function if appropriate for your application.

**Compatibility** The Palm OS 2.0 comparison function (`comparF`) has this prototype:

```
Int comparF (VoidPtr, VoidPtr, Long other);
```

The Palm OS 1.0 comparison function (`comparF`) has this prototype:

```
Int comparF (BytePtr A, BytePtr B, Long other);
```

**See Also** [SysQSort](#)

## SysKeyboardDialog

**Purpose** Pop up the system keyboard if there is a field object with the focus. The field object's text chunk is edited directly.

## **Declared In** Keyboard.h

**Prototype** void SysKeyboardDialog (KeyboardType kbd)

**Result** Returns nothing. Changes the field's text chunk.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

**See Also** [SysKeyboardDialogV10](#), [FrmSetFocus](#)

# SysKeyboardDialogV10

**Purpose** Pop up the system keyboard if there is a field object with the focus. The field object's text chunk is edited directly.

**Declared In** Keyboard.h

**Prototype** void SysKeyboardDialogV10 ()

**Parameters** None.

**Result** Returns nothing. The field's text chunk is changed.

**Compatibility** Corresponds to the 1.0 implementation of `SysKeyboardDialog`.

**See Also** SysKeyboardDialog, FrmSetFont

## SysLibFind

<b>Purpose</b>	Return a reference number for a library that is already loaded, given its name.				
<b>Declared In</b>	SystemMgr.h				
<b>Prototype</b>	Err SysLibFind (const Char *nameP, UInt16 *refNumP)				
<b>Parameters</b>	<table><tr><td>nameP</td><td>Pointer to the name of a loaded library.</td></tr><tr><td>refNumP</td><td>Pointer to a variable for returning the library reference number (on failure, this variable is undefined)</td></tr></table>	nameP	Pointer to the name of a loaded library.	refNumP	Pointer to a variable for returning the library reference number (on failure, this variable is undefined)
nameP	Pointer to the name of a loaded library.				
refNumP	Pointer to a variable for returning the library reference number (on failure, this variable is undefined)				
<b>Result</b>	0 if no error; otherwise: sysErrLibNotFound (if the library is not yet loaded), or another error returned from the library's install entry point.				
<b>Comments</b>	Most built-in libraries (NetLib, serial, IR) are preloaded automatically when the system is reset. Third-party libraries must be loaded before this call can succeed (use <a href="#">SysLibLoad</a> ). You can check if a library is already loaded by calling SysLibFind and checking for a 0 error return value (it will return a non-zero value if the library is not loaded).				

## SysLibInstall

<b>Purpose</b>	Installs a library into the Library table and calls the library's install entry point.
<b>Declared In</b>	SystemMgr.h
<b>Prototype</b>	Err SysLibInstall (SysLibEntryProcPtr libraryP, UInt16 *refNumP)
<b>Parameters</b>	-> libraryP         Pointer to the library being installed.

## System Manager

### System Functions

---

<- refNumP      Pointer to the variable in which the installed library reference number is stored upon return.

**Result**      Returns 0 if no error, memErrNotEnoughSpace if an error occurs while allocating memory in the library table, or, if an error is returned by the call to the library's install entry point, that error is returned and the reference number is set to sysInvalidRefNum.

**Comments**      This routine is largely used by the Palm OS, but can also be called by applications that need to install their own libraries.

**Compatibility**      Implemented only if [2.0 New Feature Set](#) is present.

## SysLibLoad

**Purpose**      Load a library given its database creator and type.

**Declared In**      SystemMgr.h

**Prototype**      Err SysLibLoad (UInt32 libType,  
                  UInt32 libCreator, UInt16 \*refNumP)

**Parameters**      libType      Type of library database.  
                  libCreator      Creator of library database.  
                  refNumP      Pointer to variable for returning the library reference number (on failure, sysInvalidRefNum is returned in this variable)

**Result**      0 if no error; otherwise: sysErrLibNotFound, sysErrNoFreeRAM, sysErrNoFreeLibSlots, or other error returned from the library's install entry point.

**Comments**      Presently, the "load" functionality is **not** supported when you use the Palm OS Simulator.

When an application no longer needs a library that it **successfully** loaded via SysLibLoad, it is responsible for unloading the library

by calling [SysLibRemove](#) and passing it the library reference number returned by [SysLibLoad](#). More information is available in the white paper on shared libraries, which you can find on the Palm developer support web site.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## SysLibRemove

**Purpose** Unload a library previously loaded with [SysLibLoad](#).

**Declared In** SystemMgr.h

**Prototype** Err SysLibRemove (UInt16 refNum)

**Parameters** -> refNum The library reference number.

**Result** 0 if no error; otherwise [sysErrParamErr](#) if the refNum is not a valid library reference number.

**Comments** [SysLibRemove](#) releases the dynamic memory allocated to the shared library's dispatch table, resources, and global variables.

## SysQSort

**Purpose** Sort elements in an array according to the supplied comparison function.

**Declared In** SysUtils.h

**Prototype** void SysQSort (void \*baseP, Int16 numElements, Int16 width, CmpFuncPtr comparF, Int32 other)

**Parameters** baseP Base pointer to an array of elements.

numElements Number of elements to sort (must be at least 2).

width Width of an element.

## System Manager

### *System Functions*

---

comparF      Comparison function. See Comments, below.  
other      Other data passed to the comparison function.

**Result**      Returns nothing.

**Comments**      Equal records can be in any position relative to each other because a quick sort tends to scramble the ordering of records. As a result, calling SysQSort multiple times can result in a different order if the records are not completely unique. If you don't want this behavior, use the insertion sort instead (see [SysInsertionSort](#)).

To pick the pivot point, the quick sort algorithm picks the middle of three records picked from around the middle of all records. That way, the algorithm can take advantage of partially sorted data.

These optimizations are built in:

- The routine contains its own stack to limit uncontrolled recursion. When the stack is full, an insertion sort is used because it doesn't require more stack space.
- An insertion sort is also used when the number of records is low. This avoids the overhead of a quick sort which is noticeable for small numbers of records.
- If the records seem mostly sorted, an insertion sort is performed to move only those few records that need to be moved.

In order to use SysQSort you must write a comparison function with the following prototype:

```
Int16 comparF (void *p1, void *p2, Int32 other)
```

Your comparison function must return zero if p1 equals p2, a positive integer value if p1 is greater than p2, and a negative integer value if p1 is less than p2. Note that the value of the parameter named other is passed through from the SysQSort call and can be used to control the behavior of the comparF function if appropriate for your application.

**Compatibility**      The Palm OS 2.0 comparison function (comparF) has this prototype:

```
Int comparF (VoidPtr, VoidPtr, Long other);
```

The Palm OS 1.0 comparison function (`comparF`) has this prototype:

```
Int comparF (BytePtr A, BytePtr B, Long other);
```

**See Also** [SysInsertionSort](#)

## SysRandom

**Purpose** Return a random number anywhere from 0 to `sysRandomMax`.

**Declared In** `SysUtils.h`

**Prototype** `Int16 SysRandom (Int32 newSeed)`

**Parameters** `newSeed` New seed value, or 0 to use existing seed.

**Result** Returns a random number.

## SysReset

**Purpose** Perform a soft reset and reinitialize the globals and the dynamic memory heap.

**Declared In** `SystemMgr.h`

**Prototype** `void SysReset (void)`

**Parameters** None.

**Result** No return value.

**Comments** This routine resets the system, reinitializes the globals area and all system managers, and reinitializes the dynamic heap. All database information is preserved. This routine is called when the user presses the hidden reset switch on the device.

## System Manager

### *System Functions*

---

When running an application using the simulator, this routine looks for two data files that represent the memory of card 0 and card 1. If these are found, the Palm OS memory image is created using them. If they are not found, they are created.

When running an application on the device, this routine simply looks for the memory cards at fixed locations.

## SysSetAutoOffTime

**Purpose** Set the time out value in seconds for auto-power-off. Zero means never power off.

**Declared In** SystemMgr.h

**Prototype** UInt16 SysSetAutoOffTime (UInt16 seconds)

**Parameters** seconds Time out in seconds, or 0 for no time out.

**Result** Returns previous value of time out in seconds.

## SysSetTrapAddress

**Purpose** Set the address of the function corresponding to a system trap.

**Declared In** SystemMgr.h

**Prototype** Err SysSetTrapAddress (UInt16 trapNum,  
void \*procP)

**Parameters** -> trapNum One of the routine selectors defined in  
SysTraps.h (sysTrap...) or CoreTraps.h  
on Palm OS version 3.5 and higher.

-> procP Pointer to a function that the trap identified by  
trapNum is to point to.

**Result** Returns 0 if no error, or SysErrParamErr if trapNum is greater  
than the number of traps in the trap table.

**Comments** This function is useful for patching a system trap, in combination with [SysGetTrapAddress](#). To patch a system trap in your application, first call SysGetTrapAddress to get the trap address and then save this value somewhere. Next use SysSetTrapAddress to set the trap address to point to your function. Before your application exits, remove the patch by calling SysSetTrapAddress and passing in the original trap address you saved.

---

**WARNING!** If your application patches a system trap using this function, you **must** remove the patch before your application exits. Do **not** use this mechanism to permanently patch system traps as it may cause unpredictable results for the system and other applications.

---

**Compatibility** If [5.0 New Feature Set](#) is present this function is unimplemented.

## SysStringByIndex

**Purpose** Copy a string out of a string list resource by index. String list resources are of type 'tSTL' and contain a list of strings and a prefix string.

ResEdit always displays the items in the list as starting at 1, not 0. Consider this when creating your string list.

**Declared In** SysUtils.h

**Prototype** Char \*SysStringByIndex (UInt16 resID,  
                  UInt16 index, Char \*strP, UInt16 maxLen)

**Parameters** resID                         Resource ID of the string list.  
                  index                         String to get out of the list.  
                  strP                         Pointer to space to form the string.

## System Manager

### *System Functions*

---

maxLen                   Size of strP buffer.

**Result**   Returns a pointer to the copied string. The string returned from this call will be the prefix string appended with the designated index string. Indices are 0-based; index 0 is the first string in the resource.

**Compatibility**   Implemented only if [2.0 New Feature Set](#) is present.

## SysTaskDelay

**Purpose**   Put the processor into doze mode for the specified number of ticks.

**Declared In**   SystemMgr.h

**Prototype**   Err SysTaskDelay (Int32 delay)

**Parameters**   delay                   Number of ticks to wait (see  
[SysTicksPerSecond](#))

**Result**   Returns 0 if no error.

**See Also**   [EvtGetEvent](#)

## SysTicksPerSecond

**Purpose**   Return the number of ticks per second. This routine allows applications to be tolerant of changes to the ticks per second rate in the system.

**Declared In**   SystemMgr.h

**Prototype**   UInt16 SysTicksPerSecond (void)

**Parameters**   None

**Result**   Returns the number of ticks per second.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

## SysUIAppSwitch

**Purpose** Try to make the current UI application quit and then launch the UI application specified by card number and database ID.

**Declared In** SystemMgr.h

**Prototype** Err SysUIAppSwitch (UInt16 cardNo, LocalID dbID,  
UInt16 cmd, MemPtr cmdPBP)

## System Manager

### System Functions

---

<b>Parameters</b>	-> cardNo -> dbID -> cmd -> cmdPBP	Card number for the new application; currently only card 0 is valid. ID of the new application's resource database. Action code (launch code). Action code (launch code) parameter block.
<b>Result</b>	Returns 0 if no error.	May display a fatal error message if the cardNo parameter is invalid. On debug ROMs, displays a fatal error message if there is no currently running application.
<b>Comments</b>	Do not use this function to open the system-supplied Application Launcher application. If a third-party launch is installed, you'll likely want to launch that one instead. To do this, enqueue a keyDownEvent that contains a launchChr, as shown in the section " <a href="#">Application Launcher</a> " of the user interface chapter in the <i>Palm OS Programmer's Companion</i> , vol. I. This will run whatever is run whenever you tap on the Applications icon.  If you are passing a parameter block (the cmdPBP parameter), you must set the owner of the parameter block chunk to the operating system. To do this, and for more information, see <a href="#">MemPtrSetOwner</a> . If the parameter block structure contains references by pointer or handle to any other chunks, you also must set the owner of those chunks by using <a href="#">MemHandleSetOwner</a> or <a href="#">MemPtrSetOwner</a> . If you set the owner of this parameter block properly, the system maintains the parameter block and frees it when the second application quits. If you don't set the owner of the parameter block, the system frees the parameter block as soon as the calling application quits, causing unpredictable results.	
<b>See Also</b>	<a href="#">SysAppLaunch</a> , <a href="#">Chapter 2, “Application Startup and Stop”</a> , in the <i>Palm OS Programmer's Companion</i> , vol. I.	

# Application-Defined Functions

## PilotMain

**Purpose** The entry point for all Palm OS applications, this function's sole purpose is to receive and respond to launch codes.

**Declared In** SystemMgr.h

**Prototype** UInt32 PilotMain(UInt16 cmd, void \*cmdPBP,  
UInt16 launchFlags)

**Parameters**

- > cmd	The launch code to which your application is to respond. See <a href="#">Chapter 1, “Application Launch Codes,”</a> on page 3 for a list of predefined launch codes. You may create additional launch codes; see <a href="#">“Creating Your Own Launch Codes”</a> on page 28 of the <i>Palm OS Programmer’s Companion</i> , vol. I.
---------	---

- > cmdPBP	A pointer to a structure containing any launch-command-specific parameters, or NULL if the launch code has none. See the description of each launch code for a description of the parameter structure that accompanies it, if any.
------------	--

- > launchFlags	Flags that indicate whether your application’s global variables are available, whether your application is now the active application, whether it already was the active application, and so on. See <a href="#">“Launch Flags”</a> on page 36 for a list of launch flags.
-----------------	--

**Result** Return errNone if your application processed the launch code successfully, or an appropriate error code if there was a problem. When another application invokes your application using [SysAppLaunch](#), this value is returned to the caller.

## **System Manager**

### *Application-Defined Functions*

---

**Comments** See [Chapter 2, “Application Startup and Stop,”](#) on page 19 of the *Palm OS Programmer’s Companion*, vol. I for a discussion on how applications receive and handle launch codes, with examples.

# Text Manager

---

This chapter provides information about the text manager API declared in `TextMgr.h` by discussing these topics:

- [Text Manager Data Structures](#)
- [Text Manager Functions](#)

For more information on the text manager, see the chapter “[Localized Applications](#)” on page 363 in the *Palm OS Programmer’s Companion*, vol. I.

## Text Manager Data Structures

### CharEncodingType

The `CharEncodingType` enum specifies possible character encodings. The [Character Encoding Constants](#) define the possible values for `CharEncodingType` variables.

```
UInt8 CharEncodingType;
```

A given device supports a single character encoding. The currently available devices support either the Palm™ version of Windows code page 1252 (an extension of ISO Latin 1) or the Palm version of Windows code page 932 (an extension of Shift JIS). These encodings are identical to their Windows counterparts with some additional characters added in the control range.

#### Compatibility

Prior to version 4.0, `CharEncodingType` was an enum that defined only eight character encodings. The Palm OS® 4.0 definition of `CharEncodingType` is compatible with the previous definition.

## Text Manager

### *Text Manager Functions*

---

# Text Manager Functions

## TxtByteAttr

**Purpose** Return the possible locations of a given byte within a multi-byte character.

**Declared In** TextMgr.h

**Prototype** UInt8 TxtByteAttr (UInt8 inByte)

**Parameters** -> inByte A byte representing all or part of a valid character.

**Result** Returns a byte with one or more of the following bits set:

byteAttrFirst First byte of multi-byte character.

byteAttrLast Last byte of multi-byte character.

byteAttrMiddle Middle byte of multi-byte character.

byteAttrSingle Single-byte character.

**Comments** If inByte is valid in more than one location of a character, multiple return bits are set. For example, 0x40 in the Shift JIS character encoding is valid as a single-byte character and as the low-order byte of a double-byte character. Thus, the return value for `TxtByteAttr(0x40)` on a Shift JIS system has both the `byteAttrSingle` and `byteAttrLast` bits set.

Text manager functions that need to determine the byte positioning of a character use `TxtByteAttr` to do so. You rarely need to use this function yourself.

**Compatibility** Implemented only if [International Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueByteAttr`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

## **TxtCaselessCompare**

**Purpose** Perform a case-insensitive comparison of two text buffers.

**Declared In** TextMgr.h

**Prototype** Int16 TxtCaselessCompare (const Char\* s1,  
                  UInt16 s1Len, UInt16\* s1MatchLen, const Char\* s2,  
                  UInt16 s2Len, UInt16\* s2MatchLen)

<b>Parameters</b>	-> s1	Pointer to the first text buffer to compare.
	-> s1Len	Length in bytes of the text pointed to by s1.
	<- s1MatchLen	Points to the offset of the first character in s1 that determines the sort order. Pass NULL for this parameter if you don't need to know this number.
	-> s2	Pointer to the second text buffer to compare.
	-> s2Len	Length in bytes of the text pointed to by s2.
	<- s2MatchLen	Points to the offset of the first character in s2 that determines the sort order. Pass NULL for this parameter if you don't need to know this number.

**Result** Returns one of the following values:

- < 0     If s1 occurs before s2 in alphabetical order.
- > 0     If s1 occurs after s2 in alphabetical order.
- 0       If the two substrings that were compared are equal.

**Comments** In certain character encodings (such as Shift JIS), one character may be accurately represented as either a single-byte character or a multi-byte character. `TxtCaselessCompare` accurately matches a single-byte character with its multi-byte equivalent. For this reason, the values returned in `s1MatchLen` and `s2MatchLen` are not always equal.

## Text Manager

### *Text Manager Functions*

---

You must make sure that the parameters `s1` and `s2` point to the start of a valid character. That is, they must point to the first byte of a multi-byte character or they must point to a single-byte character; if they don't, results are unpredictable.

**Compatibility** Implemented only if [International Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCaselessCompare`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

In Palm OS 4.0, the `TxtCaselessCompare` function terminates when it finds a null byte in the string. In earlier releases, it terminated only when it reached the ending byte specified by the length parameters.

**See Also** [StrCaselessCompare](#), [TxtCompare](#), [StrCompare](#)

## **TxtCharAttr**

**Purpose** Return a character's attributes.

**Declared In** `TextMgr.h`

**Prototype** `UInt16 TxtCharAttr (WChar inChar)`

**Parameters** `-> inChar` Any valid character.

**Result** Returns a 16-bit unsigned value with any of the following bits set:

`charAttrPrint` Printable

`charAttrSpace` Blank space, tab, or newline

`charAttrAlNum` Alphanumeric

`charAttrAlpha` Alphabetic

`charAttrCntrl` Control character

charAttrGraph	Character that appears on the screen; that is, is not whitespace, a control character, or a virtual character.
charAttrDelim	Word delimiter (whitespace or punctuation).

**Comments** The character passed to this function must be a valid character given the system encoding.

This function is used in the text manager's character attribute macros ([TxtCharIsAlNum](#), [TxtCharIsCntrl](#), and so on). The macros perform operations analogous to the standard C functions `isPunct`, `isPrintable`, and so on. Usually, you'd use one of these macros instead of calling `TxtCharAttr` directly.

To obtain attributes specific to a given character encoding, use [TxtCharXAttr](#).

**Compatibility** Implemented only if [International Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharAttr`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [TxtCharIsValid](#)

## **TxtCharBounds**

**Purpose** Return the boundaries of a character containing the byte at a specified offset in a string.

**Declared In** `TextMgr.h`

**Prototype** `WChar TxtCharBounds (const Char* inText,  
                  UInt32 inOffset, UInt32* outStart, UInt32* outEnd)`

**Parameters**

-> <code>inText</code>	Pointer to the text buffer to search.
-> <code>inOffset</code>	A valid offset into the buffer <code>inText</code> . This location may contain a byte in any position (start, middle, or end) of a multi-byte character.

## Text Manager

### *Text Manager Functions*

---

<- outStart      Points to the starting offset of the character containing the byte at inOffset.

<- outEnd      Points to the ending offset of the character containing the byte at inOffset.

**Result**      Returns the character located between the offsets outStart and outEnd.

**Comments**      Use this function to determine the boundaries of a character in a string or text buffer.

If the byte at inOffset is valid in more than one location of a character, the function must search back toward the beginning of the text buffer until it finds an unambiguous byte to determine the appropriate boundaries. For this reason, `TxtCharBounds` is often slow and should be used only where needed.

You must make sure that the parameter `inText` points to the beginning of the string. That is, if the string begins with a multi-byte character, `inText` must point to the first byte of that character; if it doesn't, results are unpredictable.

**Compatibility**      Implemented only if [International Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the `PalmOSGlue` library and call `TxtGlueCharBounds`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

## **TxtCharEncoding**

**Purpose** Return the minimum encoding required to represent a character.

**Declared In** TextMgr.h

**Prototype** CharEncodingType TxtCharEncoding (WChar inChar)

**Parameters** -> inChar A valid character.

**Result** A [CharEncodingType](#) value that indicates the minimum encoding required to represent inChar. If the character isn't recognizable, charEncodingUnknown is returned.

**Comments** The minimum encoding is the encoding that takes the lowest number of bytes to represent the character. For example, if the character is a blank or a tab character, the minimum encoding is charEncodingAscii because these characters can be represented in single-byte ASCII. If the character is a ü, the minimum encoding is charEncodingISO8859\_1.

Because Palm OS only supports a single character encoding at a time, the result of this function is always logically equal to or less than the encoding used on the current system. That is, you'll only receive a return value of charEncodingISO8859\_1 if you're running on a US or European system and you pass a non-ASCII character.

Use this function for informational purposes only. Your code should not assume that the character encoding returned by this function is the Palm OS system character encoding. (Instead use FtrGet as shown in the [TxtCharXAttr](#) function description.)

Use [TxtMaxEncoding](#) to determine the order of encodings.

**Compatibility** Implemented only if [International Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call

## Text Manager

### *Text Manager Functions*

---

`TxtGlueCharEncoding`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [TxtStrEncoding](#), [TxtMaxEncoding](#)

## **TxtCharIsAlNum**

**Purpose** Macro that indicates if the character is alphanumeric.

**Declared In** `TxtMgr.h`

**Prototype** `TxtCharIsAlNum (ch)`

**Parameters** `-> ch` A valid character.

**Result** Returns `true` if the character is a letter in an alphabet or a numeric digit, `false` otherwise.

**Compatibility** Valid only if [International Feature Set](#) is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsAlNum`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [TxtCharIsDigit](#), [TxtCharIsAlpha](#)

## **TxtCharIsAlpha**

**Purpose** Macro that indicates if a character is a letter in an alphabet.

**Declared In** `TxtMgr.h`

**Prototype** `TxtCharIsAlpha (ch)`

**Parameters** `-> ch` A valid character.

**Result** Returns `true` if the character is a letter in an alphabet, `false` otherwise.

**Compatibility** Valid only if [International Feature Set](#) is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsAlpha`. For more information, see [Chapter 75, “PalmOSGlue Library”](#).

**See Also** [TxtCharIsAlNum](#), [TxtCharIsLower](#), [TxtCharIsUpper](#)

## **TxtCharIsCntrl**

**Purpose** Macro that indicates if a character is a control character.

**Declared In** `TxtMgr.h`

**Prototype** `TxtCharIsCntrl (ch)`

**Parameters** `-> ch` A valid character.

**Result** Returns `true` if the character is a non-printable character, such as the bell character or a carriage return; `false` otherwise.

**Compatibility** Valid only if [International Feature Set](#) is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsCntrl`. For more information, see [Chapter 75, “PalmOSGlue Library”](#).

## **TxtCharIsDelim**

**Purpose** Macro that indicates if a character is a delimiter.

**Declared In** `TxtMgr.h`

**Prototype** `TxtCharIsDelim (ch)`

**Parameters** `-> ch` A valid character.

**Result** Returns `true` if the character is a word delimiter (whitespace or punctuation), `false` otherwise.

## Text Manager

### *Text Manager Functions*

---

**Compatibility** Valid only if [International Feature Set](#) is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsDelim`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

## **TxtCharIsDigit**

**Purpose** Macro that indicates if the character is a decimal digit.

**Declared In** `TxtMgr.h`

**Prototype** `TxtCharIsDigit (ch)`

**Parameters** `-> ch` A valid character.

**Result** Returns `true` if the character is 0 through 9, `false` otherwise.

**Compatibility** Valid only if [International Feature Set](#) is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharIsDigit`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [TxtCharIsAlNum](#), [TxtCharIsHex](#)

## **TxtCharIsGraph**

**Purpose** Macro that indicates if a character is a graphic character.

**Declared In** `TxtMgr.h`

**Prototype** `TxtCharIsGraph (ch)`

**Parameters** `-> ch` A valid character.

**Result** Returns `true` if the character is a graphic character, `false` otherwise.

<b>Comments</b>	A graphic character is any character visible on the screen, in other words, letters, digits, and punctuation marks. A blank space is not a graphic character because it is not visible.  This macro differs from <a href="#">TxtCharIsPrint</a> in that it returns <code>false</code> if the character is whitespace. <code>TxtCharIsPrint</code> returns <code>true</code> if the character is whitespace.
-----------------	---

<b>Compatibility</b>	Valid only if <a href="#">International Feature Set</a> is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call <code>TxtGlueCharIsGraph</code> . For more information, see <a href="#">Chapter 75, “PalmOSGlue Library.”</a>
----------------------	--

## **TxtCharIsHardKey**

<b>Purpose</b>	Macro that returns true if the character is one of the hard keys on the device.				
<b>Declared In</b>	<code>TxtMgr.h</code>				
<b>Prototype</b>	<code>TxtCharIsHardKey (m, ch)</code>				
<b>Parameters</b>	<table><tr><td><code>-&gt; m</code></td><td>The modifier keys from the <a href="#">keyDownEvent</a>.</td></tr><tr><td><code>-&gt; ch</code></td><td>The character from the <code>keyDownEvent</code>.</td></tr></table>	<code>-&gt; m</code>	The modifier keys from the <a href="#">keyDownEvent</a> .	<code>-&gt; ch</code>	The character from the <code>keyDownEvent</code> .
<code>-&gt; m</code>	The modifier keys from the <a href="#">keyDownEvent</a> .				
<code>-&gt; ch</code>	The character from the <code>keyDownEvent</code> .				
<b>Result</b>	<code>true</code> if the character is one of the built-in hard keys on the device, <code>false</code> otherwise.				
<b>Compatibility</b>	Valid only if <a href="#">International Feature Set</a> is present.				
<b>See Also</b>	<a href="#">ChrIsHardKey</a>				

## Text Manager

### *Text Manager Functions*

---

## **TxtCharIsHex**

<b>Purpose</b>	Macro that indicates if a character is a hexadecimal digit.
<b>Declared In</b>	<code>TxtMgr.h</code>
<b>Prototype</b>	<code>TxtCharIsHex (ch)</code>
<b>Parameters</b>	<code>-&gt; ch</code> A valid character.
<b>Result</b>	Returns <code>true</code> if the character is a hexadecimal digit from 0 to F, <code>false</code> otherwise.
<b>Compatibility</b>	Valid only if <a href="#">International Feature Set</a> is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call <code>TxtGlueCharIsHex</code> . For more information, see <a href="#">Chapter 75, “PalmOSGlue Library.”</a>
<b>See Also</b>	<a href="#">TxtCharIsDigit</a>

## **TxtCharIsLower**

<b>Purpose</b>	Macro that indicates if a character is a lowercase letter.
<b>Declared In</b>	<code>TxtMgr.h</code>
<b>Prototype</b>	<code>TxtCharIsLower (ch)</code>
<b>Parameters</b>	<code>-&gt; ch</code> A valid character.
<b>Result</b>	Returns <code>true</code> if the character is a lowercase letter, <code>false</code> otherwise.
<b>Compatibility</b>	Valid only if <a href="#">International Feature Set</a> is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call <code>TxtGlueCharIsLower</code> . For more information, see <a href="#">Chapter 75, “PalmOSGlue Library.”</a>
<b>See Also</b>	<a href="#">TxtCharIsAlpha</a> , <a href="#">TxtCharIsUpper</a>

## **TxtCharIsPrint**

<b>Purpose</b>	Macro that indicates if a character is printable.
<b>Declared In</b>	<code>TxtMgr.h</code>
<b>Prototype</b>	<code>TxtCharIsPrint (ch)</code>
<b>Parameters</b>	<code>- &gt; ch</code> A valid character.
<b>Result</b>	Returns <code>true</code> if the character is not a control character, <code>false</code> otherwise.
<b>Comments</b>	This macro differs from <a href="#">TxtCharIsGraph</a> in that it returns <code>true</code> if the character is whitespace. <code>TxtCharIsGraph</code> returns <code>false</code> if the character is whitespace. If you are using a debug ROM and you pass a virtual character to this macro, a fatal alert is generated.
<b>Compatibility</b>	Valid only if <a href="#">International Feature Set</a> is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call <code>TxtGlueCharIsPrint</code> . For more information, see <a href="#">Chapter 75, “PalmOSGlue Library.”</a>
<b>See Also</b>	<a href="#">TxtCharIsValid</a>

## Text Manager

### *Text Manager Functions*

---

## **TxtCharIsPunct**

<b>Purpose</b>	Macro that indicates if a character is a punctuation mark.
<b>Declared In</b>	<code>TxtMgr.h</code>
<b>Prototype</b>	<code>TxtCharIsPunct (ch)</code>
<b>Parameters</b>	<code>-&gt; ch</code> A valid character.
<b>Result</b>	Returns <code>true</code> if the character is a punctuation mark, <code>false</code> otherwise.
<b>Compatibility</b>	Valid only if <a href="#">International Feature Set</a> is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call <code>TxtGlueCharIsPunct</code> . For more information, see <a href="#">Chapter 75, “PalmOSGlue Library.”</a>

## **TxtCharIsSpace**

<b>Purpose</b>	Macro that indicates if a character is a whitespace character.
<b>Declared In</b>	<code>TxtMgr.h</code>
<b>Prototype</b>	<code>TxtCharIsSpace (ch)</code>
<b>Parameters</b>	<code>-&gt; ch</code> A valid character.
<b>Result</b>	Returns <code>true</code> if the character is whitespace such as a blank space, tab, or newline; <code>false</code> otherwise.
<b>Compatibility</b>	Valid only if <a href="#">International Feature Set</a> is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call <code>TxtGlueCharIsSpace</code> . For more information, see <a href="#">Chapter 75, “PalmOSGlue Library.”</a>

## **TxtCharIsUpper**

<b>Purpose</b>	Macro that indicates if a character is an uppercase letter.
<b>Declared In</b>	<code>TxtMgr.h</code>
<b>Prototype</b>	<code>TxtCharIsUpper (ch)</code>
<b>Parameters</b>	<code>- &gt; ch</code> A valid character.
<b>Result</b>	Returns <code>true</code> if the character is an uppercase letter, <code>false</code> otherwise.
<b>Compatibility</b>	Valid only if <a href="#">International Feature Set</a> is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the <code>PalmOSGlue</code> library and call <code>TxtGlueCharIsUpper</code> . For more information, see <a href="#">Chapter 75, “PalmOSGlue Library”</a> .
<b>See Also</b>	<a href="#">TxtCharIsAlpha</a> , <a href="#">TxtCharIsLower</a>

## **TxtCharIsValid**

<b>Purpose</b>	Determine whether a character is valid character given the Palm OS character encoding.
<b>Declared In</b>	<code>TextMgr.h</code>
<b>Prototype</b>	<code>Boolean TxtCharIsValid (WChar inChar)</code>
<b>Parameters</b>	<code>- &gt; inChar</code> A character.
<b>Result</b>	Returns <code>true</code> if <code>inChar</code> is a valid character; <code>false</code> if <code>inChar</code> is not a valid character.
<b>Compatibility</b>	Implemented only if <a href="#">International Feature Set</a> is present. To use this function in code intended to be run on earlier versions of Palm OS,

## Text Manager

### *Text Manager Functions*

---

link with the PalmOSGlue library and call `TxtGlueCharIsValid`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [TxtCharAttr](#), [TxtCharIsPrint](#)

## **TxtCharSize**

**Purpose** Return the number of bytes required to store the character in a string.

**Declared In** `TextMgr.h`

**Prototype** `UInt16 TxtCharSize (WChar inChar)`

**Parameters** `-> inChar` A valid character.

**Result** The number of bytes required to store the character in a string.

**Comments** Although character variables are always two-byte long `WChar` values, in some character encodings such as Shift-JIS, characters in strings are represented by a mix of one or more bytes per character. If the character can be represented by a single byte (its high-order byte is 0), it is stored in a string as a single-byte character.

**Compatibility** Implemented only if [International Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharSize`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [TxtCharBounds](#)

## TxtCharWidth

**Purpose** Return the width required to display the specified character in the current font. If the specified character does not exist within the current font, the missing character symbol is substituted.

**Declared In** TextMgr.h

**Prototype** Int16 TxtCharWidth (WChar inChar)

**Parameters** -> inChar A valid character.

**Result** Returns the width of the specified character (in pixels).

**Comments** Use [FntWCharWidth](#) or FntGlueWCharWidth instead of this routine.

**Compatibility** Implemented only if [International Feature Set](#) is present.

## TxtCharXAttr

**Purpose** Return the extended attribute bits for a character.

**Declared In** TextMgr.h

**Prototype** UInt16 TxtCharXAttr (WChar inChar)

**Parameters** -> inChar A valid character.

**Result** Returns an unsigned 16-bit value with one or more extended attribute bits set. For specific return values, look in the header files that are specific to certain character encodings (CharLatin.h or CharShiftJIS.h).

**Comments** To interpret the results, you must know the character encoding being used. Use [FtrGet](#) with sysFtrNumEncoding as the feature number to determine the character encoding. This returns one of the [CharEncodingType](#) values. For example:

## Text Manager

### *Text Manager Functions*

---

```
WChar ch;
UInt16 attr;
UInt32 encoding;
...
attr = TxtCharXAttr(ch);
if (FtrGet(sysFtrCreator, sysFtrNumEncoding,
&encoding) != errNone)
    encoding = charEncodingPalmLatin;
if (encoding == charEncodingUTF8) {
```

**Compatibility** Implemented only if [International Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueCharXAttr`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [TxtCharAttr](#)

## TxtCompare

**Purpose** Performs a case-sensitive comparison of all or part of two text buffers.

**Declared In** `TextMgr.h`

**Prototype** `Int16 TxtCompare (const Char* s1, UInt16 s1Len,  
UInt16* s1MatchLen, const Char* s2, UInt16 s2Len,  
UInt16* s2MatchLen)`

**Parameters**

-> s1	Pointer to the first text buffer to compare.
-> s1Len	The length in bytes of the text pointed to by <code>s1</code> .
<- s1MatchLen	Points to the offset of the first character in <code>s1</code> that determines the sort order. Pass <code>NULL</code> for this parameter if you don't need to know this number.
-> s2	Pointer to the second text buffer to compare.
-> s2Len	The length in bytes of the text pointed to by <code>s2</code> .

<- s2MatchLen      Points to the offset of the first character in s2 that determines the sort order. Pass NULL for this parameter if you don't need to know this number.

**Result**      Returns one of the following values:

- < 0      If s1 occurs before s2 in alphabetical order.
- > 0      If s1 occurs after s2 in alphabetical order.
- 0      If the two substrings that were compared are equal.

**Comments**      This function performs a case-sensitive comparison. If you want to perform a case-insensitive comparison, use [TxtCaselessCompare](#).

The s1MatchLen and s2MatchLen parameters are not as useful for the `TxtCompare` function as they are for the `TxtCaselessCompare` function because `TxtCompare` implements a multi-pass sort algorithm. (See the Compatibility section below for further details.) For example, comparing the string "celery" with the string "Cauliflower" returns a positive value to indicate that "celery" sorts after "Cauliflower," and it returns a match length of 1 to indicate that the second letter determines the sort order ("e" comes after "a"). However, because `TxtCompare` ultimately does a case-sensitive comparison, comparing the string "c" to the string "C" produces a negative result and a match length of 0.

In certain character encodings (such as Shift JIS), one character may be accurately represented as either a single-byte character or a multi-byte character. `TxtCompare` accurately matches a single-byte character with its multi-byte equivalent. For this reason, the values returned in s1MatchLen and s2MatchLen are not always equal.

You must make sure that the parameters s1 and s2 point to the start of a valid character. That is, they must point to the first byte of a multi-byte character or they must point to a single-byte character; if they don't, results are unpredictable.

**Compatibility**      Implemented only if [International Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS,

## **Text Manager**

### *Text Manager Functions*

---

link with the PalmOSGlue library and call `TxtGlueCompare`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

Prior to Palm OS 4.0, `TxtCompare` and [`StrCompare`](#) only performed one level of comparison and returned as soon as they found two unequal characters. For example, if you compared the string “celery” with the string “Cauliflower,” both functions returned a value indicating that “celery” should appear before “Cauliflower” because they sorted “c” before “C.”

In Palm OS 4.0, `StrCompare` calls `TxtCompare`, and `TxtCompare` performs a comparison using up to six comparison tables for sorting with increasing precision. As a result, in Palm OS 4.0 and higher, `TxtCompare` sorts “Cauliflower” before “celery.” The `TxtGlueCompare` function uses a two-pass sort on pre-4.0 devices, which will also sort “Cauliflower” before “celery.”

Palm OS 4.0 sorting of Shift-JIS characters attempts to duplicate the sorting algorithm described by the JIS standard.

In Palm OS 4.0, the `TxtCompare` function terminates when it finds a null byte in the string. In earlier releases, it terminated only when it reached the ending byte specified by the length parameters.

**See Also** [StrCompare](#), [TxtFindString](#)

## **TxtConvertEncoding**

**Purpose** Convert a text buffer from one character encoding to another.

**Declared In** TextMgr.h

**Prototype** Err TxtConvertEncoding (Boolean newConversion,  
TxtConvertStateType \*ioStateP,  
const Char \*srcTextP, UInt16 \*ioSrcBytes,  
CharEncodingType srcEncoding, Char \*dstTextP,  
UInt16 \*ioDstBytes, CharEncodingType dstEncoding,  
const Char \*substitutionStr,  
UInt16 substitutionLen)

<b>Parameters</b>	-> newConversion  <-> ioStateP  -> srcTextP  <-> ioSrcBytes  -> srcEncoding	true if this function call is starting a new conversion, or false if this function call is a continuation of a previous conversion.  If newConversion is false, this parameter must point to the same data used for the previous invocation. If newConversion is true and no subsequent calls are planned, this parameter can be NULL.  A pointer to a text buffer. If newConversion is true, this must point to the start of a text buffer. If newConversion is false, it may point to a location in the middle of a text buffer. In either case, it must point to an inter-character boundary.  A pointer to the length in bytes of the text in srcTextP that needs to be converted. Upon return, contains the number of bytes successfully processed.  The character encoding that srcTextP currently uses. This should be one of the <a href="#">Character Encoding Constants</a> .
-------------------	---	---

## Text Manager

### *Text Manager Functions*

---

<- dstTextP	A pointer to the destination text buffer or NULL. This should always point to the location where <code>TxtConvertEncoding</code> can begin writing.
<-> ioDstBytes	A pointer to the length in bytes of <code>dstTextP</code> . Upon return, contains the number of bytes required to represent <code>srcTextP</code> in the new encoding.
-> dstEncoding	The character encoding to which to convert <code>srcTextP</code> . This should be one of the <a href="#">Character Encoding Constants</a> .
-> substitutionStr	A string that should be used to substitute any invalid or invertible characters that occur in <code>srcTextP</code> . This string must already be in the destination encoding. If NULL, the function immediately returns when an invalid character is encountered.
-> substitutionLen	The number of bytes in <code>substitutionStr</code> , not including the terminating null byte.

**Result** Returns `errNone` upon success or one of the following if an error occurs:

`txtErrConvertOverflow`

The destination buffer is not large enough to contain the converted text.

`txtErrConvertUnderflow`

The end of the source buffer contains a partial character.

`txtErrNoCharMapping`

The device does not contain a mapping between the source and destination encodings for at least one of the characters in `srcTextP`.

**txtErrUnknownEncoding**

One of the specified encodings is not valid. Currently, both the source and destination encodings must match either the device's encoding or one of the Unicode character encodings.

**Comments**

This function converts `ioSrcBytes` of text in `srcTextP` from the `srcEncoding` to the `dstEncoding` character encoding and returns the result in `dstTextP`.

Currently, the focus of `TxtConvertEncoding` is to convert between Unicode-encoded text and the device's character encoding. For this reason, `TxtConvertEncoding` can only handle conversions between the device's encoding and one of UTF-8, UCS-2, UTF-16LE, or UTF-16BE. If you specify any other character encoding for either the source or the destination buffer, the error code `txtErrUnknownEncoding` is returned.

You can retrieve the device's encoding using the following function:

```
FtrGet(sysFtrCreator, sysFtrNumEncoding,  
      &encoding)
```

If you're converting text that was received from the Internet, the encoding's name is passed along with the text data. Use the [`TxtNameToEncoding`](#) function to convert the name to a `CharEncodingType` value.

The `dstTextP` buffer must be large enough to hold the result of converting `srcTextP` to the specified encoding. You can pass `NULL` for the `dstTextP` parameter to determine the required length of the buffer before actually doing the conversion. (The required length is returned in `ioDstBytes`.)

If the function encounters an invertible character in the source text, it puts `substitutionStr` in the destination buffer in that character's place and continues the conversion. When the conversion is complete, it returns `txtErrNoCharMapping` to indicate that an error occurred. If `substitutionStr` is `NULL`, the function stops the conversion and immediately returns `txtErrNoCharMapping`. `ioSrcBytes` is set to the offset of the invertible character, `dstTextP` contains the converted string up

## Text Manager

### *Text Manager Functions*

---

to that point, and `ioDstBytes` contains the size of the converted text. You can examine the character at `ioSrcBytes` and choose to move past it and continue the conversion. Follow the rules for making repeated calls to `TxtConvertEncoding` as described in the next paragraph.

You can make repeated calls to `TxtConvertEncoding` in a loop if you only want to convert part of the input buffer at a time. When you make repeated calls to this function, the first call should use `true` for `newConversion`, and `srcTextP` should point to the start of the text buffer. All subsequent calls should use the following values:

<code>newConversion</code>	<code>false</code> .
<code>ioStateP</code>	The same data that was returned by the previous invocation.
<code>srcTextP</code>	The location where this call should begin converting. Typically, this would be the previous <code>srcTextP</code> plus the number of bytes returned in <code>ioSrcBytes</code> .
	If you are skipping over an inconvertible character, <code>srcTextP</code> must point to the character after that location.
<code>&lt;-&gt; ioSrcBytes</code>	The number of bytes that this function call should convert.
<code>dstTextP</code>	A pointer to a location where this function can begin writing the converted string. You might choose to have each function call write to a different destination buffer. To have successive calls write to the same buffer, pass the previous <code>dstTextP</code> plus the number of bytes returned in <code>ioDstBytes</code> each time.
<code>ioDstBytes</code>	The number of bytes available for output in the <code>dstTextP</code> buffer. In other words, the number of bytes remaining.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## **TxtEncodingName**

**Purpose** Obtain a character encoding's name.

**Declared In** `TextMgr.h`

**Prototype** `const Char* TxtEncodingName  
(CharEncodingType inEncoding)`

**Parameters** `-> inEncoding` One of the values from [CharEncodingType](#), indicating a character encoding.

**Result** A constant string containing the name of the encoding. The possible return values are defined in `PalmLocale.h`. They are:

<code>encodingNameAscii</code>	<code>us ascii</code>
<code>encodingNameISO8859_1</code>	<code>ISO-8859-1</code>
<code>encodingNameCP1252</code>	<code>ISO-8859-1-Windows-3.1-Latin-1</code>
<code>encodingNameShiftJIS</code>	<code>Shift_JIS</code>
<code>encodingNameCP932</code>	<code>Windows-31J</code>
<code>encodingNameUTF8</code>	<code>UTF-8</code>
" "	The encoding is not known

**Comments** Use this function to obtain the official name of the character encoding, suitable to pass to an Internet application or any other application that requires the character encoding's name to be passed along with the data.

**Compatibility** Implemented only if [International Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the `PalmOSGlue` library and call `TxtGlueEncodingName`. For more information, see [Chapter 75, "PalmOSGlue Library."](#)

**See Also** [TxtNameToEncoding](#)

## Text Manager

### *Text Manager Functions*

---

## TxtFindString

**Purpose** Perform a case-insensitive search for a string in another string.

**Declared In** TextMgr.h

**Prototype** Boolean TxtFindString (const Char\* inSourceStr,  
const Char\* inTargetStr, UInt32\* outPos,  
UInt16\* outLength)

**Parameters**

- > inSourceStr Pointer to the string to be searched.
- > inTargetStr Prepared version of the string to be found. This string should either be passed directly from the strToFind field in the [sysAppLaunchCmdFind](#) launch code's parameter block or it should be prepared using the PalmOSGlue function [TxtGluePrepFindString](#).
- <- outPos Pointer to the offset of the match in inSourceStr.
- <- outLength Pointer to the length in bytes of the matching text.

**Result** Returns true if the function finds inTargetStr within inSourceStr; false otherwise.

If found, the values pointed to by the outPos and outLength parameters are set to the starting offset and the length of the matching text. If not found, the values pointed to by outPos and outLength are set to 0.

The search that TxtFindString performs is locale-dependent. On most ROMs with Latin-based encodings, TxtFindString returns true only if the string is at the beginning of a word. On Shift-JIS encoded ROMs, TxtFindString returns true if the string is located anywhere in the word.

**Comments** Use this function instead of [FindStrInStr](#) to support the global system find facility. This function contains an extra parameter,

outLength, to specify the length of the text that matched. Pass this value to [FindSaveMatch](#) in the appCustom parameter. Then when your application receives sysAppLaunchCmdGoTo, the matchCustom field contains the length of the matching text. You use the length of matching text to highlight the match within the selected record.

You must make sure that the parameters inSourceStr and inTargetStr point to the start of a valid character. That is, they must point to the first byte of a multi-byte character, or they must point to a single-byte character; if they don't, results are unpredictable.

<b>Compatibility</b>	Implemented only if <a href="#">International Feature Set</a> is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call <a href="#">TxtGlueFindString</a> . For more information, see <a href="#">Chapter 75, “PalmOSGlue Library.”</a>
----------------------	---

**See Also** [TxtCaselessCompare](#)

## TxtGetChar

**Purpose** Retrieve the character starting at the specified offset within a text buffer.

**Declared In** TextMgr.h

**Prototype** WChar TxtGetChar (const Char\* inText,  
                  UInt32 inOffset)

**Parameters** -> inText         Pointer to the text buffer to be searched.  
                  -> inOffset         A valid offset into the buffer inText. This offset must point to an inter-character boundary.

**Result** Returns the character at inOffset in inText.

**Comments** You must make sure that the parameter inText points to the start of a valid character. That is, it must point to the first byte of a multi-

## Text Manager

### *Text Manager Functions*

---

byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

**Compatibility** Implemented only if [International Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueGetChar`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [TxtGetNextChar](#), [TxtSetNextChar](#)

## **TxtGetNextChar**

**Purpose** Retrieve the character starting at the specified offset within a text buffer.

**Declared In** `TextMgr.h`

**Prototype** `UInt16 TxtGetNextChar (const Char* inText,  
                  UInt32 inOffset, WChar* outChar)`

**Parameters**

<code>-&gt; inText</code>	Pointer to the text buffer to be searched.
<code>-&gt; inOffset</code>	A valid offset into the buffer <code>inText</code> . This offset must point to an inter-character boundary.
<code>&lt;- outChar</code>	The character at <code>inOffset</code> in <code>inText</code> . Pass <code>NULL</code> for this parameter if you don't need the character returned.

**Result** Returns the size in bytes of the character at `inOffset`. If `outChar` is not `NULL` upon entry, it points to the character at `inOffset` upon return.

**Comments** You can use this function to iterate through a text buffer character-by-character in this way:

```
UInt16 i = 0;  
WChar ch;  
while (i < bufferLength) {
```

```
i += TxtGetNextChar(buffer, i, &ch);  
//do something with ch.  
}
```

You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

**Compatibility** Implemented only if [International Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueGetNextChar`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [TxtGetChar](#), [TxtGetPreviousChar](#), [TxtSetNextChar](#)

## **TxtGetPreviousChar**

**Purpose** Retrieve the character before the specified offset within a text buffer.

**Declared In** `TextMgr.h`

**Prototype** `UInt16 TxtGetPreviousChar (const Char* inText,  
UInt32 inOffset, WChar* outChar)`

**Parameters**

-> <code>inText</code>	Pointer to the text buffer to be searched.
-> <code>inOffset</code>	A valid offset into the buffer <code>inText</code> . This offset must point to an inter-character boundary.
<- <code>outChar</code>	The character immediately preceding <code>inOffset</code> in <code>inText</code> . Pass <code>NULL</code> for this parameter if you don't need the character returned.

**Result** Returns the size in bytes of the character preceding `inOffset` in `inText`. If `outChar` is not `NULL` upon entry, then it points to the character preceding `inOffset` upon return. Returns 0 if `inOffset` is at the start of the buffer (that is, `inOffset` is 0).

## Text Manager

### *Text Manager Functions*

---

**Comments** You can use this function to iterate through a text buffer character-by-character in this way:

```
WChar ch;  
/* Find the start of the character containing  
the last byte. */  
TxtCharBounds (buffer, bufferLength - 1,  
&start, &end);  
i = start;  
while (i > 0) {  
    i -= TxtGetPreviousChar(buffer, i, &ch);  
    //do something with ch.  
}
```

This function is often slower to use than [TxtGetNextChar](#) because it must determine the appropriate character boundaries if the byte immediately before the offset is valid in more than one location (start, middle, or end) of a multi-byte character. To do this, it must work backwards toward the beginning of the string until it finds an unambiguous byte.

You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

### Compatibility

Implemented only if [International Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueGetPreviousChar`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

## **TxtGetTruncationOffset**

<b>Purpose</b>	Return the appropriate byte position for truncating a text buffer such that it is at most a specified number of bytes long.
<b>Declared In</b>	<code>TextMgr.h</code>
<b>Prototype</b>	<code>UInt32 TxtGetTruncationOffset (const Char* inText,                           UInt32 inOffset)</code>
<b>Parameters</b>	<code>-&gt; inText</code> Pointer to a text buffer. <code>-&gt; inOffset</code> An offset into the buffer <code>inText</code> .
<b>Result</b>	Returns the appropriate byte offset for truncating <code>inText</code> at a valid inter-character boundary. The return value may be less than or equal to <code>inOffset</code> .
<b>Comments</b>	You must make sure that the parameter <code>inText</code> points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.
<b>Compatibility</b>	Implemented only if <a href="#">International Feature Set</a> is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call <code>TxtGlueGetTruncationOffset</code> . For more information, see <a href="#">Chapter 75, “PalmOSGlue Library.”</a>

## **TxtGetWordWrapOffset**

<b>Purpose</b>	Locate an appropriate place for a line break in a text buffer.
<b>Declared In</b>	<code>TextMgr.h</code>
<b>Prototype</b>	<code>UInt32 TxtGetWordWrapOffset (const Char *iTextP,                           UInt32 iOffset)</code>
<b>Parameters</b>	<code>-&gt; iTextP</code> Pointer to a text buffer.

## Text Manager

### *Text Manager Functions*

---

-> iOffset	Pointer to the offset where the search should begin. The search is performed backward starting from this offset.
<b>Result</b>	Returns the offset of a character that can begin on a new line (typically, the beginning of the word that contains iOffset or last word before iOffset). If an appropriate break could not be found, returns iOffset.
<b>Comments</b>	The <a href="#">FntWordWrap</a> and <a href="#">FntWordWrapReverseNLines</a> functions call <code>TxtGetWordWrapOffset</code> to locate an appropriate place to break the text. The returned offset points to the character that should begin the next line.  This function starts at iOffset and works backward until it finds a character that typically occurs between words (for example, white space or punctuation). Then it moves forward until it locates the character that begins a word (typically, a letter or number). Note that this function may return an offset value that is greater than the one passed in if the offset passed in occurs immediately before white space or in the middle of white space.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TxtWordBounds</a>

## **TxtMaxEncoding**

<b>Purpose</b>	Return the higher of two encodings.
<b>Declared In</b>	<code>TextMgr.h</code>
<b>Prototype</b>	<code>CharEncodingType TxtMaxEncoding (CharEncodingType a, CharEncodingType b)</code>
<b>Parameters</b>	<code>-&gt; a</code> A character encoding to compare.

- > b                          Another character encoding to compare.

**Result** Returns the higher of a or b. One character encoding is higher than another if it is more specific. For example code page 1252 is “higher” than ISO 8859-1 because it represents more characters than ISO 8859-1.

**Comments** This function is used by [TxtStrEncoding](#) to determine the encoding required for a string.

**Compatibility** Implemented only if [International Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueMaxEncoding`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [TxtCharEncoding](#), [CharEncodingType](#)

## **TxtNameToEncoding**

**Purpose** Return an encoding’s constant given its name.

**Declared In** `TextMgr.h`

**Prototype** `CharEncodingType TxtNameToEncoding  
(const Char *iEncodingName)`

**Parameters** -> `iEncodingName`  
One of the string constants containing the official name of an encoding. See [TxtEncodingName](#) for a list.

**Result** Returns one of the [Character Encoding Constants](#). Returns `charEncodingUnknown` if the specified encoding could not be found.

**Comments** Use this function to convert a character encoding’s name as received from an Internet application into the character encoding constant that some text manager functions require.

## Text Manager

### *Text Manager Functions*

---

This function properly converts aliases for a character encoding. For example, passing the strings "us-ascii", "ASCII", "cp367", and "IBM367" all return `charEncodingAscii`.

The known character encodings are device-dependent. For example, a device with the Shift-JIS encoding will not know all of the aliases for Latin character encodings; however, it will know all of the aliases for Shift-JIS.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TxtEncodingName](#)

## TxtNextCharSize

**Purpose** Macro that returns the size of the character starting at the specified offset within a text buffer.

**Declared In** `TxtMgr.h`

**Prototype** `TxtNextCharSize (inText, inOffset)`

**Parameters** `-> inText` Pointer to the text buffer to be searched.  
`-> inOffset` A valid offset into the buffer `inText`. This offset must point to an inter-character boundary.

**Result** Returns (as a `UInt16`) the size in bytes of the character at `inOffset`.

**Comments** You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

**Compatibility** Valid only if [International Feature Set](#) is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with

the PalmOSGlue library and call `TxtGlueNextCharSize`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [TxtGetNextChar](#)

## **TxtParamString**

**Purpose** Replace substrings within a string with the specified values.

**Declared In** `TextMgr.h`

**Prototype** `Char* TxtParamString (const Char* inTemplate,  
const Char* param0, const Char* param1,  
const Char* param2, const Char* param3)`

**Parameters**

-> <code>inTemplate</code>	The string containing the substrings to replace.
-> <code>param0</code>	String to replace ^0 with or NULL.
-> <code>param1</code>	String to replace ^1 with or NULL.
-> <code>param2</code>	String to replace ^2 with or NULL.
-> <code>param3</code>	String to replace ^3 with or NULL.

**Result** Returns a pointer to a locked relocatable chunk in the dynamic heap that contains the appropriate substitutions.

**Comments** This function searches `inTemplate` for occurrences of the sequences ^0, ^1, ^2, and ^3. When it finds these, it replaces them with the corresponding string passed to this function. Multiple instances of each sequence will be replaced.

The replacement strings can also contain the substitution strings, provided they refer to a later parameter. That is, the `param0` string contain have references to ^1, ^2, and ^3, the `param1` string can have references to ^2 and ^3, and the `param2` string can have references to ^3. Any other occurrences of the substitution strings in the replacement strings are ignored. For example, if `param3` is the string “^0”, any occurrences of ^3 in `inTemplate` are replaced with the string “^0”.

## Text Manager

### *Text Manager Functions*

---

You must make sure that the parameter `inTemplate` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

`TxtParamString` allocates space for the returned string in the dynamic heap through a call to `MemHandleNew`, and then returns the result of calling `MemHandleLock` with this handle. Your code is responsible for freeing this memory when it is no longer needed.

**Compatibility** Implemented if [3.5 New Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the `PalmOSGlue` library and call `TxtGlueParamString`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [TxtReplaceStr](#), [FrmCustomAlert](#)

## **TxtPreviousCharSize**

**Purpose** Macro that returns the size of the character before the specified offset within a text buffer.

**Declared In** `TxtMgr.h`

**Prototype** `TxtPreviousCharSize (inText, inOffset)`

**Parameters** `-> inText` Pointer to the text buffer.  
`-> inOffset` A valid offset into the buffer `inText`. This offset must point to an inter-character boundary.

**Result** Returns (as a `UInt16`) the size in bytes of the character preceding `inOffset` in `inText`. Returns 0 if `inOffset` is at the start of the buffer (that is, `inOffset` is 0).

**Comments** You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

This macro is often slower to use than [TxtNextCharSize](#) because it must determine the appropriate character boundaries if the byte immediately before the offset is valid in more than one location (start, middle, or end) of a multi-byte character. To do this, it must work backwards toward the beginning of the string until it finds an unambiguous byte.

**Compatibility** Valid only if [International Feature Set](#) is present. To use this macro in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGluePreviousCharSize`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [TxtGetPreviousChar](#)

## **TxtReplaceStr**

**Purpose** Replace a substring of a given format with another string.

**Declared In** `TextMgr.h`

**Prototype** `UInt16 TxtReplaceStr (Char* ioStr,  
                  UInt16 inMaxLen, const Char* inParamStr,  
                  UInt16 inParamNum)`

**Parameters** `<-> ioStr` The string in which to perform the replacing.  
`-> inMaxLen` The maximum length in bytes that `ioStr` can become.  
`-> inParamStr` The string that `^inParamNum` should be replaced with. If `NULL`, no changes are made.  
`-> inParamNum` A single-digit number (0 to 9).

**Result** Returns the number of occurrences found and replaced.  
Returns a fatal error message if `inParamNum` is greater than 9.

**Comments** This function searches `ioStr` for occurrences of the string `^inParamNum`, where `inParamNum` is any digit from 0 to 9. When it finds the string, it replaces it with `inParamStr`. Multiple instances

## Text Manager

### *Text Manager Functions*

---

will be replaced as long as the resulting string doesn't contain more than `inMaxLen` bytes, not counting the terminating null.

You can set the `inParamStr` parameter to NULL to determine the required length of `ioStr` before actually doing the replacing. `TxtReplaceStr` returns the number of occurrences it finds of `^inParamNum`. Multiply this value by the length of the `inParamStr` you intend to use to determine the appropriate length of `ioStr`.

You must make sure that the parameter `ioStr` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

<b>Compatibility</b>	Implemented only if <a href="#">International Feature Set</a> is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call <code>TxtGlueReplaceStr</code> . For more information, see <a href="#">Chapter 75, “PalmOSGlue Library.”</a>
----------------------	--

## **TxtSetNextChar**

**Purpose** Set a character within a text buffer.

**Declared In** `TextMgr.h`

**Prototype** `UInt16 TxtSetNextChar (Char* ioText,  
                  UInt32 inOffset, WChar inChar)`

<b>Parameters</b>	<code>&lt;-&gt; ioText</code>	Pointer to a text buffer.
	<code>-&gt; inOffset</code>	A valid offset into the buffer <code>inText</code> . This offset must point to an inter-character boundary.
	<code>-&gt; inChar</code>	The character to replace the character at <code>inOffset</code> with. Must not be a virtual character.

**Result** Returns the size of `inChar`.

**Comments** This function replaces the character in `ioText` at the location `inOffset` with the character `inChar`. Note that there must be enough space at `inOffset` to write the character.

You can use [TxtCharSize](#) to determine the size of `inChar`.

You must make sure that the parameter `ioText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

**Compatibility** Implemented only if [International Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueSetNextChar`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [TxtGetNextChar](#)

## **TxtStrEncoding**

**Purpose** Return the encoding required to represent a string.

**Declared In** `TextMgr.h`

**Prototype** `CharEncodingType TxtStrEncoding  
(const Char* inStr)`

**Parameters** `-> inStr` A string.

**Result** A [CharEncodingType](#) value that indicates the encoding required to represent `inChar`. If any character in the string isn't recognizable, then `charEncodingUnknown` is returned.

**Comments** The encoding for the string is the maximum encoding of any character in that string. For example, if a two-character string contains a blank space and a ü, the appropriate encoding is `charEncodingISO8859_1`. The blank space's minimum encoding is ASCII. The minimum encoding for the ü is ISO 8859-1. The maximum of these two encodings is ISO 8859-1.

## Text Manager

### *Text Manager Functions*

---

Because Palm OS only supports a single character encoding at a time, the results of this function is always logically equal to or less than the encoding used on the current system. That is, you'll only receive a return value of `charEncodingISO8859_1` if you're running on a Latin-based system.

Use this function for informational purposes only. Your code should not assume that the character encoding returned by this function is the Palm OS system's character encoding. (Instead use `FtrGet` as shown in the [TxtCharXAttr](#) function description.)

**Compatibility** Implemented only if [International Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueStrEncoding`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [TxtCharEncoding](#), [TxtMaxEncoding](#)

## **TxtTransliterate**

**Purpose** Converts the specified number of bytes in a text buffer using the specified operation.

**Declared In** `TextMgr.h`

**Prototype** `Err TxtTransliterate (const Char* inSrcText,  
                  UInt16 inSrcLength, Char* outDstText,  
                  UInt16* ioDstLength, TranslitOpType inOp)`

**Parameters**

-> <code>inSrcText</code>	Pointer to a text buffer.
-> <code>inSrcLength</code>	The length in bytes of <code>inSrcText</code> .
<- <code>outDstText</code>	The output buffer containing the converted characters.
<- <code>ioDstLength</code>	Upon entry, the maximum length of <code>outDstText</code> . Upon return, the actual length of <code>outDstText</code> .

-> inOp	A 16-bit unsigned value that specifies which transliteration operation is to be performed. The values possible for this field are specific to the character encoding used on a particular device. These operations are universally available:
translitOpUpperCase	Converts the character to uppercase letters.
translitOpLowerCase	Converts the characters to lowercase letters.
translitOpPreprocess	Don't actually perform the operation. Instead, return in <code>ioDstLength</code> the amount of space required for the output text.

**Result** Returns one of the following values:

errNone	Success
txtErrUnknownTranslitOp	inOp's value is not recognized
txtErrTranslitOverrun	inSrcText and outDstText point to the same memory location and the operation has caused the function to overwrite unprocessed data in the input buffer.
txtErrTranslitOverflow	outDstText is not large enough to contain the converted string.
txtErrTranslitUnderflow	The end of the source buffer contains a partial character.

**Comments** inSrcText and outDstText may point to the same location if you want to perform the operation in place. However, you should

## Text Manager

### *Text Manager Functions*

---

be careful that the space required for `outDstText` is not larger than `inSrcText` so that you don't generate a `txtErrTranslitOverrun` error.

For example, suppose on a Shift JIS encoded system, you want to convert a series of single-byte Japanese Katakana symbols to double-byte Katakana symbols. You cannot perform this operation in place because it replaces a single-byte character with a multi-byte character. When the first converted character is written to the buffer, it overwrites the second input character. Thus, a text overrun has occurred.

You can ensure that you have enough space for the output by OR-ing your chosen operation with `translitOpPreprocess`. For example, to convert a string to uppercase letters, do the following:

```
outSize = buf2Len;
error = TxtTransliterate(buf1, buf1len, &buf2,
&outSize,
translitOpUpperCase|translitOpPreprocess);
if (outSize > buf2len)
    /* allocate more memory for buf2 */
error = TxtTransliterate(buf1, buf1Len, &buf2,
&outSize, translitOpUpperCase);
```

You must make sure that the parameter `inSrcText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

#### Compatibility

Implemented only if [International Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueTransliterate`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

In Palm OS 4.0, the `TxtTransliterate` function terminates when it finds a null byte in the source string. In earlier releases, it terminated only when it reached the ending byte specified by the length parameter.

## **TxtWordBounds**

**Purpose** Find the boundaries of a word of text that contains the character starting at the specified offset.

**Declared In** TextMgr.h

**Prototype** Boolean TxtWordBounds (const Char\* inText,  
                  UInt32 inLength, UInt32 inOffset,  
                  UInt32\* outStart, UInt32\* outEnd)

**Parameters**

-> inText	Pointer to a text buffer.
-> inLength	The length in bytes of the text pointed to by inText.
-> inOffset	A valid offset into the text buffer inText. This offset must point to the beginning of a character.
<- outStart	The starting offset of the text word.
<- outEnd	The ending offset of the text word.

**Result** Returns `true` if a word is found. Returns `false` if the word doesn't exist or is punctuation or whitespace.

**Comments** Assuming the ASCII encoding, if the text buffer contains the string "Hi! How are you?" and you pass 5 as the offset, `TxtWordBounds` returns the start and end of the word containing the character at offset 5, which is the character "o". Thus, `outStart` and `outEnd` would point to the start and end of the word "How".

You must make sure that the parameter `inText` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character; if it doesn't, results are unpredictable.

**Compatibility** Implemented only if [International Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `TxtGlueWordBounds`. For more information, see [Chapter 75, "PalmOSGlue Library."](#)

## **Text Manager**

### *Text Manager Functions*

---

In Palm OS 4.0, the `TxtWordBounds` function terminates when it finds a null byte in the string. In earlier releases, it terminated only when it reached the ending byte specified by the length parameter.

**See Also** [`TxtCharBounds`](#), [`TxtCharIsDelim`](#), [`TxtGetWordWrapOffset`](#)

# Text Services Manager

---

This chapter provides information about the Text Services Manager API as declared in `TextServicesMgr.h`. The Text Services Manager provides the caller with an API for interacting with various text services, including front-end processors (FEPs), which are sometimes known as input methods. This chapter discusses the following topics:

- [Text Services Manager Data Structures](#)
- [Text Services Manager Functions](#)

## Text Services Manager Data Structures

### TsmFepModeType

The `TsmFepModeType` type specifies the input modes used by the functions [TsmGetFepMode](#) and [TsmSetFepMode](#).

```
typedef UInt16 TsmFepModeType;
```

`TsmFepModeType` can be set to one of the defined modes listed in the following table.

Constant	Value	Description
<code>tsmFepModeDefault</code>	0	The default input mode for the FEP. For example, with the Japanese FEP, the default mode is Hiragana.

## Text Services Manager

### *Text Services Manager Functions*

---

Constant	Value	Description
tsmFepModeOff	1	Indicates that there is no active FEP input mode (the FEP is off).
tsmFepModeCustom	128	A custom FEP input mode. You can have more than one custom mode; the starting value is 128. Katagana is an example of a custom input mode for the Japanese FEP.

## Text Services Manager Functions

### TsmGetFepMode

**Purpose** Return the current input mode for the active front-end processor (FEP).

**Declared In** TextServicesMgr.h

**Prototype** TsmFepModeType TsmGetFepMode (void \*nullParam)

**Parameters** -> nullParam An unused status pointer that must be set to NULL.

**Result** If there is an active FEP, returns the current mode for the active FEP. If there is no active FEP, returns tsmFepModeOff.

**Comments** The most common use for this function is to save the current FEP mode. You could then call [TsmSetFepMode](#) to set the current mode to “off” and again to restore the saved mode once the application has finished using a special text field.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present. In Palm OS 3.5, the nullParam parameter takes a non-NULL value, allowing the caller to maintain its own status record. In Palm OS 4.0, this

parameter is unused and must be set to NULL. Any other value generates a non-fatal alert.

**See Also** [TsmSetFepMode](#)

## TsmSetFepMode

**Purpose** Set the input mode for the active front-end processor (FEP) to be the mode defined by the parameter `inNewMode`.

**Declared In** `TextServicesMgr.h`

**Prototype** `TsmFepModeType TsmSetFepMode (void *nullParam,  
TsmFepModeType inNewMode)`

**Parameters** `-> nullParam` An unused status pointer that must be set to NULL.

`-> inNewMode` The new FEP input mode.

**Result** Returns the previous input mode. If there is no active FEP, returns `tsmFepModeOff`.

**Comments** The most common use for this function is to set the FEP mode to "off" while the application is using a special text field, and then to restore the previous mode. See [TsmGetFepMode](#) for more information on saving and restoring the FEP mode.

One common reason for explicitly disabling the FEP in code is when a text field will only contain 7-bit ASCII (numeric fields automatically turn off the FEP). For example, if the application has a password field and the contents of that field will always be 7-bit ASCII, the application should turn off the FEP to help prevent the user from entering invalid characters into the field.

Another common case occurs when the application has a numeric field, but cannot just rely on the numeric field attribute. For example, if you want the user to be able to enter the minus ("−") sign, you cannot use a numeric field because the field code prevents the user from entering this character: its not a digit or a period. In this case, you should make it a regular field and have the

## Text Services Manager

### *Text Services Manager Functions*

---

application screen the characters. The application should disable the FEP when such a pseudo-numeric field is active.

---

**IMPORTANT:** A mode change is currently enqueued as a keyDown event so that the field and FEP can remain properly synchronized and so that the mode change doesn't affect any pending keyDown events. The mode change does not happen until the enqueued keyDown event is passed to `FrmHandleEvent`; if you call [TsmGetFepMode](#) immediately after calling [TsmSetFepMode](#), you won't see a mode change.

There are also some current limitations with changing the mode: there must be an active form; and if there is an active field in the form, it must not be a numeric field.

---

#### **Compatibility**

Implemented only if [3.5 New Feature Set](#) is present. In Palm OS 3.5, the `nullParam` parameter takes a non-NULL value, allowing the caller to maintain its own status record. In Palm OS 4.0, this parameter is unused and must be set to NULL. Any other value generates a non-fatal alert.

#### **See Also**

[TsmGetFepMode](#), [EvtEnqueueKey](#), [FrmHandleEvent](#)

# Time Manager

---

This chapter provides reference material for the time manager.

- [Time Manager Data Structures](#)
- [Time Manager Functions](#)

The header file `DateTime.h` declares the API that this chapter describes. For more information on the time manager, see the section “[Time](#)” in the *Palm OS Programmer’s Companion*, vol. I.

## Time Manager Data Structures

The time manager uses these structures to store information.

### DateFormatType

The `DateFormatType` enum specifies the different display formats for date values.

```
typedef enum {
    dfMDYWithSlashes,
    dfDMYWithSlashes,
    dfDMYWithDots,
    dfDMYWithDashes,
    dfYMDWithSlashes,
    dfYMDWithDots,
    dfYMDWithDashes,
    dfMDYLongWithComma,
    dfDMYLong,
    dfDMYLongWithDot,
    dfDMYLongNoDay,
    dfDMYLongWithComma,
    dfYMDLongWithDot,
    dfYMDLongWithSpace,
    dfMYMed,
    dfMYMedNoPost,
```

## Time Manager

### Time Manager Data Structures

---

```
    dfMDYWithDashes  
} DateFormatType;
```

#### Value Descriptions

dfMDYWithSlashes	The month, day, and year numbers separated by slashes. For example, 12/31/95.  This is considered a short format.
dfDMYWithSlashes	The day, month, and year numbers separated by slashes. For example, 31/12/95.  This is considered a short format.
dfDMYWithDots	The day, month, and year numbers separated by dots. For example, 31.12.95.  This is considered a short format.
dfDMYWithDashes	The day, month, and year numbers separated by dashes. For example, 31-12-95.  This is considered a short format.
dfYMDWithSlashes	The year, month, and day numbers separated by slashes. For example, 95/12/31.  This is considered a short format.
dfYMDWithDots	The year, month, and day numbers separated by dots. For example, 95.12.31.  This is considered a short format.
	This is considered a short format.

dfYMDWithDashes	The year, month, and day numbers separated by dashes. For example, 95-12-31.
	This is considered a short format.
dfMDYLongWithComma	The month, day, and year in long format, with a comma. For example, Dec 31, 1995.
	This is considered a long format.
dfDMYLong	The month, day, and year in long format. For example, 31 Dec 1995.
	This is considered a long format.
dfDMYLongWithDot	The month, day, and year in long format, with a dot. For example, 31. Dec 1995.
	This is considered a long format.
dfDMYLongNoDay	The month and year in long format. For example, Dec 1995.
	This is considered a long format.
dfDMYLongWithComma	The day, month, and year in long format, with a comma. For example, 31 Dec, 1995.
	This is considered a long format.
dfYMDLongWithDot	The year, month, and day in long format with dot separators. For example, 1995.12.31.
	This is considered a long format.

## Time Manager

### Time Manager Data Structures

---

dfYMDLongWithSpace	The year, month, and day in long format with space separators. For example, 1995 Dec 31.
dfMYMed	This is considered a long format.
dfMYMedNoPost	The month in long format with the two-digit year, preceded by an apostrophe. For example, Dec '95.
dfMDYWithDashes	This is considered a medium format.
	The month, day, and year numbers separated by dashes. For example, 12-31-95.
	This is considered a short format.

#### Compatibility

The dfMDYWithDashes constant is defined in Palm OS® 4.0 and higher.

## DateType

The DateType structure represents a date and time value.

```
typedef struct {
    Int16    second;
    Int16    minute;
    Int16    hour;
    Int16    day;
    Int16    month;
    Int16    year;
    Int16    weekDay;
} DateType
typedef DateType *DatePtr;
```

### Field Descriptions

second     The number of seconds. This is a value between 0 and 59.  
minute    The number of minutes. This is a value between 0 and 59.  
hour      The number of hours. This is a value between 0 and 23.  
day       The day number. This is a value between 1 and 31.  
month     The month number. This is a value between 1 and 12.  
year      The year number.  
weekDay    The day number. This represents the number of days since Sunday and is thus a value between 0 and 6.

## DateType

The DateType structure represents a date value.

```
typedef struct {
    UInt16 year    :7;
    UInt16 month   :4;
    UInt16 day     :5;
} DateType;

typedef DateType *DatePtr;
```

### Field Descriptions

year      The number of years since 1904.  
            Note that this is the format used on Macintosh computers.  
month     The month number. This is a value between 1 and 12.  
day       The day number. This is a value between 1 and 31.

## DaylightSavingsTypes

The DaylightSavingsTypes enum specifies the different forms of daylight savings times that you can specify for date and time values.

## Time Manager

### Time Manager Data Structures

---

Note that the table uses “DST” to represent daylight savings time.

```
typedef enum {
    dsNone,
    dsUSA,
    dsAustralia,
    dsWesternEuropean,
    dsMiddleEuropean,
    dsEasternEuropean,
    dsGreatBritain,
    dsRumania,
    dsTurkey,
    dsAustraliaShifted
} DaylightSavingsTypes;
```

#### Value Descriptions

dsNone	No DST (daylight savings time)
dsUSA	U.S.A. DST
dsAustralia	Australian DST
dsWesternEuropean	Western European DST
dsMiddleEuropean	Middle European DST
dsEasternEuropean	Eastern European DST
dsGreatBritain	Great Britain and Eire DST
dsRumania	Rumanian DST
dsTurkey	Turkish DST
dsAustraliaShifted	Australian DST, with the 1986 shift

#### Compatibility

If [4.0 New Feature Set](#) is present, this data type is obsolete. In versions 4.0 and higher, Palm OS represents daylight savings time as an integer value that gives the number of minutes to add to the current time for daylight savings time.

## DayOfMonthType

The DayOfMonth enum specifies the different day-of-the-week numeric values that are returned by the [DayOfMonth](#) function. These values are used to represent repeating appointments that occur on specific days of the month; for example, the first Friday or the third Tuesday of each month.

```
typedef enum {
    dom1stSun, dom1stMon, dom1stTue, dom1stWen,
    dom1stThu, dom1stFri, dom1stSat,
    dom2ndSun, dom2ndMon, dom2ndTue, dom2ndWen,
    dom2ndThu, dom2ndFri, dom2ndSat,
    dom3rdSun, dom3rdMon, dom3rdTue, dom3rdWen,
    dom3rdThu, dom3rdFri, dom3rdSat,
    dom4thSun, dom4thMon, dom4thTue, dom4thWen,
    dom4thThu, dom4thFri, dom4thSat,
    domLastSun, domLastMon, domLastTue,
    domLastWen, domLastThu, domLastFri,
    domLastSat
} DayOfWeekType;
```

### Value Descriptions

dom1stSun	The first Sunday of the month.
dom1stMon	The first Monday of the month.
dom1stTue	The first Tuesday of the month.
dom1stWen	The first Wednesday of the month.
dom1stThu	The first Thursday of the month.
dom1stFri	The first Friday of the month.
dom1stSat	The first Saturday of the month.
dom2ndSun	The second Sunday of the month.
dom2ndMon	The second Monday of the month.
dom2ndTue	The second Tuesday of the month.
dom2ndWen	The second Wednesday of the month.

## Time Manager

### *Time Manager Data Structures*

---

dom2ndThu	The second Thursday of the month.
dom2ndFri	The second Friday of the month.
dom2ndSat	The second Saturday of the month.
dom3rdSun	The third Sunday of the month.
dom3rdMon	The third Monday of the month.
dom3rdTue	The third Tuesday of the month.
dom3rdWen	The third Wednesday of the month.
dom3rdThu	The third Thursday of the month.
dom3rdFri	The third Friday of the month.
dom3rdSat	The third Saturday of the month.
dom4thSun	The fourth Sunday of the month.
dom4thMon	The fourth Monday of the month.
dom4thTue	The fourth Tuesday of the month.
dom4thWen	The fourth Wednesday of the month.
dom4thThu	The fourth Thursday of the month.
dom4thFri	The fourth Friday of the month.
dom4thSat	The fourth Saturday of the month.
domLastSun	The last Sunday of the month.
domLastMon	The last Monday of the month.
domLastTue	The last Tuesday of the month.
domLastWen	The last Wednesday of the month.
domLastThu	The last Thursday of the month.
domLastFri	The last Friday of the month.
domLastSat	The last Saturday of the month.

**Compatibility** On Palm OS versions earlier than 4.0, this type was named DayOfWeekType.

## TimeFormatType

The TimeFormatType enum specifies the different display formats for time values.

```
typedef enum
{
    tfColon,
    tfColonAMPM,
    tfColon24h,
    tfDot,
    tfDotAMPM,
    tfDot24h,
    tfHoursAMPM,
    tfHours24h,
    tfComma24h,
} TimeFormatType;

typedef TimeFormatType *TimeFormatPtr;
```

### Value Descriptions

tfColon	The hour and minutes separated by a colon character. For example, 1:00.
tfColonAMPM	The hour and minutes separated by a colon and followed by an AM/PM indication. For example, 1:00 pm.
tfColon24h	The 24-hour time with the hour and minutes separated by a colon character. For example, 13:00.
tfDot	The hour and minutes separated by a dot character. For example, 1.00.
tfDotAMPM	The hour and minutes separated by a period and followed by an AM/PM indication. For example, 1.00 pm.

## Time Manager

### Time Manager Constants

---

tfDot24h	The 24-hour time with the hour and minutes separated by a dot character. For example, 13 . 00.
tfHoursAMPM	The hour value followed by an AM/PM indication. For example, 1 pm.
tfHours24h	The 24-hour value, followed by an AM/PM indication. For example, 13.
tfComma24h	The 24-hour time with the hour and minutes separated by a comma character. For example, 13 , 00.

## TimeType

The TimeType structure represents a time value.

```
typedef struct {
    UInt8    hours;
    UInt8    minutes;
} TimeType;
typedef TimeType *TimePtr;
```

### Field Descriptions

hours The number of hours. This is a value between 0 and 23.

minutes The number of minutes. This is value between 0 and 59.

## Time Manager Constants

The following table shows the constants that represent the maximum lengths of strings returned by the date and time formatting routines [DateToAscii](#), [DateToDOWDMFormat](#), and [TimeToAscii](#).

<b>Constant</b>	<b>Value</b>	<b>Description</b>
dateStringLength	9	Maximum length of the string returned by <code>DateToAscii</code> for short date formats.
longDateStrLength	15	Maximum length of the string returned by <code>DateToAscii</code> for medium and long date formats.
timeStringLength	9	Maximum length of the string returned by <code>TimeToAscii</code> .
dowDateStringLength	19	Maximum length of the string returned by <code>DateToDoWDMFormat</code> for short date formats.
dowLongDateStrLength	25	Maximum length of the string returned by <code>DateToDoWDMFormat</code> for both medium and long date formats.

## Time Manager Functions

### DateAdjust

**Purpose** Return a new date +/- the days adjustment.

**Declared In** `DateTime.h`

**Prototype** `void DateAdjust (DatePtr dateP, Int32 adjustment)`

<b>Parameters</b>	<code>&lt;-&gt; dateP</code>	A pointer to a <a href="#">DateType</a> structure with the date to be adjusted.
	<code>-&gt; adjustment</code>	The number of days by which to adjust the date.

**Result** Returns nothing. Upon return, `dateP` contains the adjusted date.

## Time Manager

### Time Manager Functions

---

**Comments** This function advances the date and manages month and year wrapping conditions.

**See Also** [TimAdjust](#)

## DateDaysToDate

**Purpose** Converts a date specified as the number of days since January 1, 1904 to a [DateType](#) structure.

**Declared In** DateTime.h

**Prototype** void DateDaysToDate (UInt32 days, DatePtr date)

**Parameters** -> days The number of days since 1/1/1904.  
<-> date A pointer to a [DateType](#) structure that is updated with the computed date values.

**Result** Returns nothing. Upon return, the date information is returned in the structure referenced by the date parameter.

**See Also** [DateSecondsToDate](#), [DateToDays](#)

## DateSecondsToDate

**Purpose** Converts a date specified as the number of seconds since January 1, 1904 to a [DateType](#) structure.

**Declared In** DateTime.h

**Prototype** void DateSecondsToDate (UInt32 seconds, DatePtr date)

**Parameters** -> seconds The number of seconds since 1/1/1904.

<- date A pointer to a [DateType](#) structure that is updated with the computed date values.

**Result** Returns nothing. The structure referenced by the date parameter is updated with the date information.

**See Also** [DateDaysToDate](#), [DateToDays](#)

## DateTemplateToAscii

**Purpose** Convert the specified date values into a string that is formatted according to a formatting template specification.

**Declared In** `DateTime.h`

**Prototype** `UInt16 DateTemplateToAscii (const Char *templateP,  
                  UInt8 months, UInt8 days, UInt16 years,  
                  Char *stringP, Int16 stringLen)`

**Parameters** -> `templateP` A pointer to the template string used to format the date.

See the Comments section below for details on how to specify date formatting in this template string.

-> `months` The month number, which must be a value between 1 and 12.

-> `days` The day number, which must be a value between 1 and 31.

-> `years` The four-digit year number. For example, 1995.

<- `stringP` A pointer to a string that is updated with the result.

If `stringP` is NULL, this function does not write an output string; however, it does return the length required for the output string.

## Time Manager

### Time Manager Functions

---

If `stringP` is not `NULL`, this function writes the formatted string to `stringP`, writing up to `stringSize` bytes into `stringP`.

-> `stringLen` The size of the `stringP` buffer.

**Result** The length of the formatted string, without the terminating null byte.

The `DateTemplateToAscii` returns the required length of the formatted string even if the `stringP` parameter is `NULL`, this allows you to determine the buffer size at runtime.

**Comments** This function is intended as a replacement for the `DateToAscii` and `DateToDOWDMFormat` functions.

This function uses the formatting template referenced by `templateP` to create a formatted string from the date values that you pass in.

You specify a series of formatting substrings in `templateP`. Each substring has the form:

---

^<valueType><formatModifier>

---

Each substring has three components:

- The ^ character begins a substring.
- The <valueType> component is a single-digit value that specifies the value type.
- The <formatModifier> component is a single-letter value that specifies how you want that value formatted.

The following is an example of a template specification with three substrings:

^0z ^21 ^4r

[Table 52.1](#) shows the values you can specify for the <valueType> component. Note that the formatted result depends on the <modifier> value.

**Table 52.1 Template value types for the DateTemplateToAscii function**

Value	Value type	Formatted examples
0	Day number	1, 01, 23, 31
1	Day name	Tue, Tuesday
2	Month name	May, Aug, August
3	Month number	4, 04, 11
4	Year number	97, 1997

[Table 52.2](#) shows the values you can specify for the <modifier> component of each template substring.

**Table 52.2 Template modifier types for the DateTemplateToAscii function**

Modifier	Description
s	Formats the value in short form
r	Formats the value in regular form
l	Formats the value in long form
z	Adds a leading zero to the formatted numeric value

Finally, [Table 52.3](#) shows examples of each value type formatted with each modifier type.

**Table 52.3 Examples of formatted values**

Value type	Raw value	s (Short format)	r (Regular format)	l (Long format)	z (Zero format)
0 (Day number)	2	2	2	2	02
1 (Day name)	2	T	Tue	Tuesday	n/a

## Time Manager

### Time Manager Functions

---

**Table 52.3 Examples of formatted values (*continued*)**

Value type	Raw value	s (Short format)	r (Regular format)	l (Long format)	z (Zero format)
2 <b>(Month name)</b>	11	N	Nov	November	n/a
3 <b>(Month number)</b>	11	11	11	11	11
4 <b>(Year number)</b>	2000	00	2000	2000	n/a

For example, calling `DateTemplateToAscii` as follows:

```
DateTemplateToAscii("^0z ^21 ^4r", 2, 7,  
2000, myStr, 20)
```

Produces the following formatted string:

```
07 February 2000
```

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call `DateGlueTemplateToAscii`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [DateToAscii](#), [DateToDOWDMFormat](#)

## DateToAscii

<b>Purpose</b>	Convert the passed date to a string using the format specified by the dateFormat parameter.										
<b>Declared In</b>	DateTime.h										
<b>Prototype</b>	<pre>void DateToAscii (UInt8 months, UInt8 days,                   UInt16 years, DateFormatType dateFormat,                   Char *pString)</pre>										
<b>Parameters</b>	<table><tr><td>-&gt; months</td><td>The month number, which must be a value between 1 and 12.</td></tr><tr><td>-&gt; days</td><td>The day number, which must be a value between 1 and 31.</td></tr><tr><td>-&gt; years</td><td>The four-digit year number. For example, 1995.</td></tr><tr><td>-&gt; dateFormat</td><td>Any <a href="#">DateFormatType</a> format.</td></tr><tr><td>&lt;- pString</td><td>A pointer to string that is updated with the result.  This string must be of length dateStringLength for short formats or longDateStrLength for medium or long formats. Note that these lengths do include the terminating null byte. For more information about required string lengths, see <a href="#">Time Manager Constants</a>.</td></tr></table>	-> months	The month number, which must be a value between 1 and 12.	-> days	The day number, which must be a value between 1 and 31.	-> years	The four-digit year number. For example, 1995.	-> dateFormat	Any <a href="#">DateFormatType</a> format.	<- pString	A pointer to string that is updated with the result.  This string must be of length dateStringLength for short formats or longDateStrLength for medium or long formats. Note that these lengths do include the terminating null byte. For more information about required string lengths, see <a href="#">Time Manager Constants</a> .
-> months	The month number, which must be a value between 1 and 12.										
-> days	The day number, which must be a value between 1 and 31.										
-> years	The four-digit year number. For example, 1995.										
-> dateFormat	Any <a href="#">DateFormatType</a> format.										
<- pString	A pointer to string that is updated with the result.  This string must be of length dateStringLength for short formats or longDateStrLength for medium or long formats. Note that these lengths do include the terminating null byte. For more information about required string lengths, see <a href="#">Time Manager Constants</a> .										
<b>Result</b>	Returns nothing. The string reference by pString is updated with the formatted string.										
<b>Comments</b>	If you are using a debug ROM, the string buffer is filled with either dateStringLength or longStrLength debugging bytes, depending on the value of the dateFormat parameter.  It is important to allocate enough space for your string buffer. Finding buffer overflow errors can be difficult when using a debug ROM. One common situation is when you pass a buffer that is too										

## Time Manager

### Time Manager Functions

---

small from a form, for an element such as a label or title. Then, the buffer overflow can clobber objects that follow the form in memory. When a form element's location information is corrupted, it disappears from the display.

Note that you can use the [DateTemplateToAscii](#) function instead of this function if the 3.5 feature set is present. You can call the [DateTemplateToAscii](#) function with a NULL string buffer to predetermine the required size for your buffer.

**See Also** [TimeToAscii](#), [DateToDOWDMFormat](#), [DateTemplateToAscii](#)

## DateToDays

**Purpose** Convert the [DateType](#) structure to the number of days elapsed from January 1, 1904.

**Declared In** DateTime.h

**Prototype** UInt32 DateToDays (DateType date)

**Parameters** -> date A [DateType](#) structure.

**Result** Returns the number of days elapsed from January 1, 1904 to the specified date.

**See Also** [DateDaysToDate](#)

## DateToDOWDMFormat

**Purpose** Convert a date to a formatted string using the format specified by the dateFormat parameter. The resultant string includes the name of the day of the week.

**Declared In** `DateTime.h`

**Prototype** `void DateToDOWDMFormat (UInt8 months, UInt8 days,  
UInt16 years, DateFormatType dateFormat,  
Char *pString)`

**Parameters**

-> months	The month number, which must be a value between 1 and 12.
-> days	The day number, which must be a value between 1 and 31.
-> years	The four-digit year number. For example, 1995.
-> dateFormat	Any <a href="#">DateFormatType</a> format.
<- pString	A pointer to a string that is updated with the result. The string must be of length <code>dowDateStringLength</code> for short formats or <code>dowLongDateStrLength</code> for medium or long date formats. See <a href="#">Time Manager Constants</a> for string buffer lengths.

**Result** Returns nothing. The string referenced by `pString` is updated with the formatted string.

**Comments** The values of some of the [Time Manager Constants](#) that specify the required string buffer lengths do change from time to time. You should always use the constants or verify the required lengths by checking the `datetime.h` file.

It is important to allocate enough space for your string buffer. Finding buffer overflow errors can be difficult when using a debug ROM. One common situation is when you pass a buffer that is too small from a form, for an element such as a label or title. Then, the

## Time Manager

### Time Manager Functions

---

buffer overflow can clobber objects that follow the form in memory. When a form element's location information is corrupted, it disappears from the display.

Note that you can use the [DateTemplateToAscii](#) function instead of this function if the 3.5 feature set is present. You can call the [DateTemplateToAscii](#) function with a NULL string buffer to predetermine the required size for your buffer.

<b>Compatibility</b>	On Palm OS 3.1 Japanese ROMs, this function contains a bug that prevented it from properly displaying 4-byte long day names. To prevent this bug from affecting your application, use <code>DateGlueToDOWDMFormat</code> in the <code>PalmOSGlue</code> library instead of calling this function directly. For more information, see <a href="#">Chapter 75, "PalmOSGlue Library."</a>
----------------------	--

<b>See Also</b>	<a href="#">DateToAscii</a> , <a href="#">DateTemplateToAscii</a>
-----------------	---

## DayOfMonth

<b>Purpose</b>	Return a value that represents the day of a month on which the specified date occurs. The value represents a quantity such as "First Monday" or "Third Friday" as is used for repeating appointments in the Datebook.
----------------	---

<b>Declared In</b>	<code>DateTime.h</code>
--------------------	-------------------------

<b>Prototype</b>	<code>Int16 DayOfMonth (Int16 month, Int16 day, Int16 year)</code>
------------------	--

<b>Parameters</b>	<code>-&gt; month</code>	The month number, which must be a value between 1 and 12.
	<code>-&gt; day</code>	The day number, which must be a value between 1 and 31.
	<code>-&gt; year</code>	The four-digit year number. For example, 1995.

<b>Result</b>	Returns a value that represents day of the month. This value is one of the <a href="#">DayOfMonthType</a> values.
---------------	---

**Comments** The returns value can be used to specify on which day of the month an appointment repeats.

## DayOfWeek

**Purpose** Return the day of the week value for a specified date.

**Declared In** `DateTime.h`

**Prototype** `Int16 DayOfWeek (Int16 month, Int16 day,  
Int16 year)`

**Parameters**

-> <code>month</code>	The month number, which must be a value between 1 and 12.
-> <code>day</code>	The day number, which must be a value between 1 and 31.
-> <code>year</code>	The four-digit year number. For example, 1995.

**Result** Returns one of the following values for the day of the week of the specified date, as shown in the following table:

<b>Day name</b>	<b>Returned day value</b>
Sunday	0
Monday	1
Tuesday	2
Wednesday	3
Thursday	4
Friday	5
Saturday	6

## Time Manager

### *Time Manager Functions*

---

## DaysInMonth

**Purpose** Return the number of days in the month.

**Declared In** `DateTime.h`

**Prototype** `Int16 DaysInMonth (Int16 month, Int16 year)`

**Parameters**

<code>-&gt; month</code>	The month number, which must be a value between 1 and 12.
<code>-&gt; year</code>	The four-digit year number. For example, 1995.

**Result** Returns the number of days in the month for the specified year.

## TimAdjust

**Purpose** Return a new date, with the time adjusted by the specified number of seconds.

**Declared In** `DateTime.h`

**Prototype** `void TimAdjust (DateTimePtr dateTimeP, Int32 adjustment)`

**Parameters**

<code>&lt;-&gt; dateTimeP</code>	A pointer to a <a href="#">DateType</a> structure.
<code>-&gt; adjustment</code>	The number of seconds by which to adjust the time.

**Result** Returns nothing. The structure referenced by `dateTimeP` is modified to contain the updated date and time.

**Comments** This function advances the time by the specified number of seconds and takes care of any wraparound conditions.

**See Also** [DateAdjust](#)

## TimDateTimeToSeconds

**Purpose** Return the number of seconds elapsed from 12:00 A.M. on January 1, 1904 to the specified date and time.

**Declared In** DateTime.h

**Prototype** UInt32 TimDateTimeToSeconds  
(DateTimePtr dateTimeP)

**Parameters** -> dateTimeP A pointer to a [DateTimeType](#) structure.

**Result** The number of seconds elapsed from 12:00 A.M. on January 1, 1904 to the date referenced by dateTimeP.

**See Also** [TimSecondsToDateTime](#)

## TimGetSeconds

**Purpose** Return the current date and time of the device in seconds since 12:00 A.M. on January 1, 1904.

**Declared In** TimeMgr.h

**Prototype** UInt32 TimGetSeconds (void)

**Parameters** None.

**Result** The number of seconds elapsed from 12:00 A.M. on January 1, 1904 to the current date and time on the device.

**See Also** [TimSetSeconds](#)

## Time Manager

### Time Manager Functions

---

## TimGetTicks

**Purpose** Return the tick count since the last reset. The tick count does not advance while the device is in sleep mode.

**Declared In** TimeMgr.h

**Prototype** UInt32 TimGetTicks (void)

**Parameters** None.

**Result** Returns the tick count.

**Comments** You can call the [SysTicksPerSecond](#) routine to determine the number of ticks per second.

**See Also** [SysTicksPerSecond](#)

## TimSecondsToDateTime

**Purpose** Converts a date specified as the number of seconds since January 1, 1904 to a [DateTimeType](#) structure.

**Declared In** DateTime.h

**Prototype** void TimSecondsToDateTime (UInt32 seconds,  
DateTimePtr dateTImeP)

**Parameters** -> seconds      A date specified as the number of seconds elapsed from 12:00 A.M. on January 1, 1904 to the date

<- `dateTimeP` A pointer to a [DateTimeType](#) structure that is updated with the date and time values.

**Result** Returns nothing. The structure referenced by `dateTimeP` is updated with the date and time computed for the number of seconds since 12:00 A.M. on January 1, 1904.

**See Also** [TimDateTimeToSeconds](#)

## TimSetSeconds

**Purpose** Set the clock of the device to the date and time passed as the number of seconds since 12:00 A.M. on January 1, 1904.

**Declared In** TimeMgr.h

**Prototype** `void TimSetSeconds (UInt32 seconds)`

**Parameters** `-> seconds` The number of seconds since 12:00 A.M. on January 1, 1904.

**Result** Returns nothing.

**Comments** If the [Notification Feature Set](#) is present, this function broadcasts the `sysNotifyTimeChangeEvent` to all interested parties. See [Chapter 39, “Notification Manager,”](#) for more information.

**See Also** [TimGetSeconds](#)

## Time Manager

### Time Manager Functions

---

## TimeToAscii

<b>Purpose</b>	Convert the time to a string that is formatted according to the specified <code>timeFormat</code> .	
<b>Declared In</b>	<code>DateTime.h</code>	
<b>Prototype</b>	<code>void TimeToAscii (UInt8 hours, UInt8 minutes, TimeFormatType timeFormat, Char *pString)</code>	
<b>Parameters</b>	<code>-&gt; hours</code>	The number of hours. This must be a value between 0 and 23.
	<code>-&gt; minutes</code>	The number of minutes. This must be a value between 0 and 59.
	<code>-&gt; timeFormat</code>	The time format for the resultant string. This must be one of the <a href="#">TimeFormatType</a> values.
	<code>&lt;- pString</code>	A pointer to a string that is updated with the resultant string. This string must be of length <code>timeStringLength</code> .  See <a href="#">Time Manager Constants</a> for information on string buffer lengths.
<b>Result</b>	Returns nothing. The string referenced by <code>pString</code> is updated with the formatted string.	
<b>Comments</b>	<p>If you are using a debug ROM in Palm OS 3.5, the string buffer is filled with <code>timeStringLength</code> debugging bytes.</p> <p>It is important to allocate enough space for your string buffer. Finding buffer overflow errors can be difficult when using a debug ROM. One common situation is when you pass a buffer that is too small from a form, for an element such as a label or title. Then, the buffer overflow can clobber objects that follow the form in memory. When a form element's location information is corrupted, it disappears from the display.</p>	
<b>See Also</b>	<a href="#">DateToAscii</a>	

## TimeZoneToAscii

**Purpose** Convert a time zone to a string.

**Declared In** DateTime.h

**Prototype** void TimeZoneToAscii (Int16 timeZone,  
const LmLocaleType \*localeP, Char \*string)

**Parameters**

-> timeZone	A pointer to the time zone, given as minutes east of Greenwich Mean Time (GMT).
-> localeP	A pointer to a locale (see <a href="#">LmLocaleType</a> ) that identifies the time zone country. You can use the constant <code>lmAnyLanguage</code> as the value for the language field of the structure pointed to by this parameter.
<- string	A pointer to a string in which to return the result. This string must be of length <code>timeZoneStringLength</code> .

**Result** Returns nothing.

**Comments** This function returns a descriptive string for the specified time zone. This string identifies the time zone first by its country, such as "USA (Mountain)" or "Canada (Eastern)." If the function cannot find a time zone that matches the specified GMT offset and country, it returns a string containing the time zone as a numeric offset from the GMT (for example, "GMT+9:00").

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## Time Manager

### Time Manager Functions

---

## TimTimeZoneToUTC

<b>Purpose</b>	Converts a date and time from a given time zone to Universal Coordinated Time (UTC). UTC is also known as Greenwich Mean Time (GMT).						
<b>Declared In</b>	DateTime.h						
<b>Prototype</b>	UInt32 TimTimeZoneToUTC (UInt32 seconds, Int16 timeZone, Int16 daylightSavingAdjustment)						
<b>Parameters</b>	<table><tr><td>-&gt; seconds</td><td>The number of seconds since 12:00 A.M. on January 1, 1904.</td></tr><tr><td>-&gt; timeZone</td><td>The time zone, given as the number of minutes east of UTC. For time zones west of UTC but before the international dateline, this is a negative number.</td></tr><tr><td>-&gt; daylightSavingAdjustment</td><td>The number of minutes to add to the current time for daylight savings time in this time zone.</td></tr></table>	-> seconds	The number of seconds since 12:00 A.M. on January 1, 1904.	-> timeZone	The time zone, given as the number of minutes east of UTC. For time zones west of UTC but before the international dateline, this is a negative number.	-> daylightSavingAdjustment	The number of minutes to add to the current time for daylight savings time in this time zone.
-> seconds	The number of seconds since 12:00 A.M. on January 1, 1904.						
-> timeZone	The time zone, given as the number of minutes east of UTC. For time zones west of UTC but before the international dateline, this is a negative number.						
-> daylightSavingAdjustment	The number of minutes to add to the current time for daylight savings time in this time zone.						
<b>Result</b>	Returns the same time as seconds but in the Universal Coordinated Time. The value is still given as the number of seconds since 12:00 A.M. on January 1, 1904.						
<b>Comments</b>	<p>The returned value is not necessarily the time in Greenwich because Greenwich may be observing daylight saving time.</p> <p>You can use this function to convert the local time to UTC. The time zone and the daylight savings adjustment are system preferences that can be retrieved using <a href="#">PrefGetPreference</a>. For example, the following code converts the current local time to UTC:</p>						

```
Int16 timeZone =  
    PrefGetPreference(prefTimeZone) ;  
Int16 daylightSavingAdjustment =  
    PrefGetPreference(  
        prefDaylightSavingAdjustment) ;  
UInt32 utcTime =
```

```
TimTimeZoneToUTC(TimGetSeconds(), timeZone,  
daylightSavingAdjustment);
```

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TimUTCToTimeZone](#)

## TimUTCToTimeZone

**Purpose** Converts a date and time from Universal Coordinated Time (UTC) to the specified time zone. UTC is also known as Greenwich Mean Time (GMT).

**Declared In** `DateTime.h`

**Prototype** `UInt32 TimUTCToTimeZone (UInt32 seconds,  
Int16 timeZone, Int16 daylightSavingAdjustment)`

**Parameters**

-> <code>seconds</code>	The number of seconds since 12:00 A.M. on January 1, 1904 in UTC.
-> <code>timeZone</code>	The time zone, given as the number of minutes east of UTC. For time zones west of UTC before the international dateline, this is a negative number.
-> <code>daylightSavingAdjustment</code>	The number of minutes to add to the current time for daylight savings time in this time zone.

**Result** Returns the same time as `seconds` but in the specified time zone. The value is still given as the number of seconds since 12:00 A.M. on January 1, 1904.

**Comments** The `seconds` value is not necessarily the time in Greenwich because Greenwich may be observing daylight saving time.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TimTimeZoneToUTC](#)

## **Time Manager**

### *Time Manager Functions*

---

# Virtual File System Manager

---

The Virtual File System (VFS) Manager is a layer of software that manages all installed file system libraries. It provides a unified API to application developers while allowing them to seamlessly access many different types of file systems—such as VFAT, HFS, and NFS—on many different types of media, including Compact Flash, Memory Stick, and SmartMedia. This chapter provides reference material for the VFS Manager API as follows:

- [VFS Manager Data Structures](#)
- [VFS Manager Constants](#)
- [VFS Manager Functions](#)
- [Application-Defined Functions](#)

The header file `VFSMgr.h` declares the VFS Manager API. For more information on the VFS Manager, see [Chapter 7, “Expansion,”](#) in *Palm OS Programmer’s Companion*, vol. I.

Note that the VFS Manager is an optional system extension; the functions described in this chapter are implemented only if the [VFS Manager Feature Set](#) is present.

## VFS Manager Data Structures

### **FileInfoType**

The `FileInfoType` structure contains information about a specified file or directory. This information is returned as a parameter to [VFSDirEntryEnumerate](#). The structure is defined as follows:

```
typedef struct FileInfoTag {  
    UInt32 attributes;
```

## **Virtual File System Manager**

### *VFS Manager Data Structures*

---

```
    Char *nameP;
    UInt16 nameBufLen;
} FileInfoType, *FileInfoPtr;
```

#### **Field Descriptions**

attributes	Characteristics of the file or directory. See <a href="#">File and Directory Attributes</a> for the bits that make up this field.
nameP	Pointer to the buffer that receives the full name of the file or directory. Initialize this parameter to NULL if you don't want to receive the name.
nameBufLen	Size of the nameP buffer, in bytes.

## **FileRef**

The `FileRef` type is used to encode references to files and directories.

```
typedef UInt32 FileRef;
```

## **VFSAnyMountParamType**

The `VFSAnyMountParamType` structure is a base structure for [`VFSSlotMountParamType`](#), [`VFSPOSEMountParamType`](#), and other similar structures that may be defined in the future. Use one or the other according to how you set the `mountClass` parameter.

```
typedef struct VFSAnyMountParamTag {
    UInt16 volRefNum;
    UInt16 reserved;
    UInt32 mountClass;
} VFSAnyMountParamType;

typedef VFSAnyMountParamType
*VFSAnyMountParamPtr;
```

### Field Descriptions

volRefNum	The volume reference number. This is initially obtained when you successfully mount a volume. It can then be used to format a volume with <a href="#">VFSVolumeFormat</a> or unmount a volume with <a href="#">VFSVolumeUnmount</a> .
reserved	Reserved for future use.
mountClass	Defines the type of mount to use with the specified volume. See <a href="#">Volume Mount Classes</a> for a list of mount types.

## VFSSlotMountParamType

The VFSSlotMountParamType structure is used when you are mounting a card located in an Expansion Manager slot. The vfsMountParam->mountClass field must be set to VFSMountClass\_SlotDriver.

```
typedef struct VFSSlotMountParamTag {
    VFSAnyMountParamType vfsMountParam;
    UInt16 slotLibRefNum;
    UInt16 slotRefNum;
} VFSSlotMountParamType;
```

### Field Descriptions

vfsMountParam	See the description of <a href="#">VFSAnyMountParamType</a> for an explanation of the fields in this structure. Set vfsMountParam->mountClass to VFSMountClass_SlotDriver to mount an Expansion Manager slot.
slotLibRefNum	Reference number for the slot driver library allocated to the given slot number. Obtain this field by calling <a href="#">ExpSlotLibFind</a> .
slotRefNum	Number of the slot to be mounted.

## VFSPOSEMountParamType

The VFSPOSEMountParamType structure is used when you are mounting a volume through the Palm OS® Emulator. The vfsMountParam->mountClass must be set to VFSMountClass\_POSE. Note that ordinary applications and file systems shouldn't use VFSPOSEMountParamType.

```
typedef struct VFSPOSEMountParamTag {  
    VFSAnyMountParamType vfsMountParam;  
    UInt8 poseSlotNum;  
} VFSPOSEMountParamType
```

### Field Descriptions

vfsMountParam	See the description of <a href="#">VFSAnyMountParamType</a> for an explanation of the fields in this structure. Set vfsMountParam->mountClass to VFSMountClass_POSE to mount a virtual slot.
poseSlotNum	Number of the virtual slot number to be mounted by the Palm OS Emulator.

## VolumeInfoType

The VolumeInfoType structure defines information that is returned to [VFSVolumeInfo](#) and used throughout the VFS functions.

```
typedef struct VolumeInfoTag {  
    UInt32 attributes;  
    UInt32 fsType;  
    UInt32 fsCreator;  
    UInt32 mountClass;  
    UInt16 slotLibRefNum;  
    UInt16 slotRefNum;  
    UInt32 mediaType;  
    UInt32 reserved;  
} VolumeInfoType, *VolumeInfoPtr;
```

## Field Descriptions

attributes	Characteristics of the volume. See <a href="#">Volume Attributes</a> for the bits that make up this field.
fsType	File system type for this volume. See <a href="#">Defined File Systems</a> for a list of the supported file systems.
fsCreator	Creator ID of this volume's file system driver. This information is used with <a href="#">VFSCustomControl</a> .
mountClass	Mount class that mounted this volume. The supported mount classes are listed under <a href="#">Volume Mount Classes</a> .
slotLibRefNum	Reference to the slot driver library with which the volume is mounted. This field is only valid when the mount class is <code>vfsMountClass_SlotDriver</code> .
slotRefNum	Slot number where the card containing the volume is loaded. This field is only valid when the mount class is <code>vfsMountClass_SlotDriver</code> .
mediaType	Type of card media. See <a href="#">Defined Media Types</a> in the <a href="#">Expansion Manager</a> chapter for the list of values. This field is only valid when the mount class is <code>vfsMountClass_SlotDriver</code> .
reserved	Reserved for future use.

## VFS Manager Constants

### Defined File Systems

The following file systems are currently defined by the VFS Manager. These values are used with [VFSVolumeInfo](#) in the `VolumeInfoType.fsType` parameter.

## **Virtual File System Manager**

### *VFS Manager Constants*

---

<b>Constant</b>	<b>Value</b>	<b>Description</b>
<code>vfsFilesystemType_AFS</code>	'afsu'	Unix Andrew file system
<code>vfsFilesystemType_EXT2</code>	'ext2'	Linux file system
<code>vfsFilesystemType_FAT</code>	'fats'	FAT12 and FAT16, which only handles 8.3 filenames
<code>vfsFilesystemType_FFS</code>	'ffsb'	Unix Berkeley block based file system
<code>vfsFilesystemType_HFS</code>	'hfss'	Macintosh standard hierarchical file system
<code>vfsFilesystemType_HFSPlus</code>	'hfse'	Macintosh extended hierarchical file system
<code>vfsFilesystemType_HPFS</code>	'hpfs'	OS2 High Performance file system
<code>vfsFilesystemType_MFS</code>	'mfso'	Macintosh original file system
<code>vfsFilesystemType_NFS</code>	'nfsu'	Unix Networked file system
<code>vfsFilesystemType_Novell</code>	'novl'	Novell file system
<code>vfsFilesystemType_NTFS</code>	'ntfs'	Windows NT file system
<code>vfsFilesystemType_VFAT</code>	'vfat'	FAT12 and FAT16 extended to handle long filenames

---

## Open Mode Constants

This section describes constants that are used for the `openMode` parameter to the [VFSFileOpen](#) function. These constants specify the mode in which a file or directory is opened.

Constant	Value	Description
<code>vfsModeExclusive</code>	0x0001U	Open and lock the file or directory. This mode excludes anyone else from using the file or directory until it is closed.
<code>vfsModeRead</code>	0x0002U	Open for read access.
<code>vfsModeWrite</code>	0x0004U   <code>vfsModeExclusive</code>	Open for write access.
<code>vfsModeReadWrite</code>	<code>vfsModeWrite</code>   <code>vfsModeRead</code>	Open for read/write access.
<code>vfsModeCreate</code>	0x0008U	Create the file if it doesn't already exist. This open mode is implemented in the VFS layer, rather than in the file system library.
<code>vfsModeTruncate</code>	0x0010U	Truncate the file to zero (0) bytes after opening, removing all existing data. This open mode is implemented in the VFS layer, rather than in the file system library.
<code>vfsModeVFLayerOnly</code>	<code>vfsModeCreate</code>   <code>vfsModeTruncate</code>	Mask used to isolate those flags that are only used by the VFS layer. These flags are not passed to the file system layer.
<code>vfsModeLeaveOpen</code>	0x0020U	Leave the file open even after the application exits.

## **Virtual File System Manager**

### *VFS Manager Constants*

---

## **File and Directory Attributes**

The constants in the following table define bits that can be used individually or in combination when setting or interpreting the file attributes for a given file or directory. See [VFSFileGetAttributes](#), [VFSFileSetAttributes](#), and the [FileInfoType](#) data structure for specific use.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
vfsFileAttrReadOnly	0x00000001UL	Read-only file or directory
vfsFileAttrHidden	0x00000002UL	Hidden file or directory
vfsFileAttrSystem	0x00000004UL	System file or directory
vfsFileAttrVolumeLabel	0x00000008UL	Volume label
vfsFileAttrDirectory	0x00000010UL	Directory
vfsFileAttrArchive	0x00000020UL	Archived file or directory
vfsFileAttrLink	0x00000040UL	Link to another file or directory

## **Volume Attributes**

The constants in the following table define bits that can be used individually or in combination to make up the attributes field in the [VolumeInfoType](#) structure.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
vfsVolumeAttrHidden	0x00000004UL	The volume should not be visible to the user.
vfsVolumeAttrReadOnly	0x00000002UL	The volume is read only.
vfsVolumeAttrSlotBased	0x00000001UL	Reserved. Check the mount class to determine how a volume is mounted.

## Volume Mount Classes

The following constants define how a given volume is mounted. The `mountClass` field in the [VFSAnyMountParamType](#) and [VolumeInfoType](#) structures takes on one of these values.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
<code>vfsMountClass_POSE</code>	'pose'	Mount the volume through the Palm OS Emulator. This is used for testing.
<code>vfsMountClass_Simulator</code>	<code>sysFileTSSimulator</code>	Mount the volume through the simulator. This is used for testing.
<code>vfsMountClass_SlotDriver</code>	<code>sysFileTSSlotDriver</code>	Mount the volume with a slot driver shared library.

## Error Codes

The VFS Manager defines the following error codes:

<b>Constant</b>	<b>Description</b>
<code>vfsErrBadData</code>	The operation could not be completed because of invalid data.
<code>vfsErrBadName</code>	Invalid filename, path, or volume label.
<code>vfsErrBufferOverflow</code>	The supplied buffer is too small.
<code>vfsErrDirectoryNotFound</code>	Returned when the path leading up to the file does not exist.
<code>vfsErrDirNotEmpty</code>	The directory is not empty and therefore cannot be deleted.

## **Virtual File System Manager**

### *VFS Manager Constants*

---

<b>Constant</b>	<b>Description</b>
vfsErrFileAlreadyExists	A file with this name exists already in this location.
vfsErrFileBadRef	The file reference is invalid: it has been closed or was not obtained from <a href="#">VFSFileOpen</a> .
vfsErrFileEOF	The file pointer is at the end of the file.
vfsErrFileGeneric	Generic file error.
vfsErrFileNotFoundException	The file was not found at the specified location.
vfsErrFilePermissionDenied	The requested permissions could not be granted.
vfsErrFileStillOpen	Returned from the underlying file system's delete function if the file is still open.
vfsErrIsADirectory	This operation can only be performed on a regular file, not a directory.
vfsErrNameShortened	A volume name or filename was automatically shortened to conform to the file system specification.
vfsErrNoFileSystem	None of the installed file systems support this operation.
vfsErrNotADirectory	This operation can only be performed on a directory.
vfsErrVolumeBadRef	The volume reference number is invalid.

Constant	Description
vfsErrVolumeFull	There is insufficient space left on the volume.
vfsErrVolumeStillMounted	Returned from the underlying file system's format function if the volume is still mounted.

## VFS Manager Functions

### VFSCustomControl

**Purpose** Make a custom API call to a particular file system, given its creator ID. You can use [VFSVolumeInfo](#) to determine the creator ID of the file system for a given volume.

**Declared In** VfsMgr.h

**Prototype** Err VFSCustomControl (UInt32 fsCreator,  
 UInt32 apiCreator, UInt16 apiSelector,  
 void \*valueP, UInt16 \*valueLenP)

<b>Parameters</b>	-> fsCreator	Creator of the file system to call. A value of zero (0) tells the VFS Manager to check each registered file system, looking for one which supports the call.
	-> apiCreator	Registered creator ID.
	-> apiSelector	Custom operation to perform.
	<-> valueP	A pointer to a buffer containing data specific to the operation. On exit, depending on the function of the particular custom call and on the value of valueLenP, the contents of this buffer may have been updated.

## **Virtual File System Manager**

### *VFS Manager Functions*

---

<-> valueLenP On entry, points to the size of the valueP buffer. On exit, this value reflects the size of the data written to the valueP buffer. If valueLenP is NULL, valueP is passed to the file system but is not updated on exit.

**Result** Returns one of the following error codes:

errNone No error.

expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.

expErrUnsupportedOperation  
The specified opcode and/or creator is unsupported or undefined.

sysErrParamErr The valueP buffer is too small.

vfsErrNoFileSystem  
VFS Manager cannot find an appropriate file system to handle the request.

**Comments** The driver identifies the call and its API by a registered creator ID and a selector. This allows file system developers to extend the API by defining selectors for their creator IDs. It also allows file system developers to support selectors (and custom calls) defined by other file system developers.

This function must return expErrUnsupportedOperation for all unsupported or undefined opcodes and/or creators.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

## **VFSDirCreate**

**Purpose** Create a new directory.

**Declared In** VfsMgr.h

**Prototype** Err VFSDirCreate(UInt16 volRefNum,  
const Char \*dirNameP)

**Parameters**

-> volRefNum	Volume reference number returned from <a href="#">VFSVolumeEnumerate</a> .
-> dirNameP	Pointer to the full path of the directory to be created.

**Result** Returns one of the following error codes:

errNone No error

expErrNotOpen The file system library necessary for this call  
has not been installed or has not been opened.

vfsErrBadName Some or all of the path, up to but not including  
the last component specified in the dirNameP  
parameter, does not exist.

vfsErrFileAlreadyExists

A file with this name already exists in this  
location.

vfsErrNoFileSystem

The VFS Manager cannot find an appropriate  
file system to handle the request.

vfsErrVolumeBadRef

The volume has not been mounted.

vfsErrVolumeFull

There is not enough space left on the volume.

**Comments** All parts of the path except the last component must already exist.  
The `vfsFileAttrDirectory` attribute is set with this function.

## **Virtual File System Manager**

### *VFS Manager Functions*

---

[VFSDirCreate](#) does not open the directory. Any operations you want to perform on this directory require a reference, which is obtained through a call to [VFSFileOpen](#).

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

## **VFSDirEntryEnumerate**

**Purpose** Enumerate the entries in a given directory. Entries can include files, links, and other directories.

**Declared In** VfsMgr.h

**Prototype** Err VFSDirEntryEnumerate (FileRef dirRef,  
                  UInt32 \*dirEntryIteratorP, FileInfoType \*infoP)

**Parameters**

-> dirRef	Directory reference returned from <a href="#">VFSFileOpen</a> .
<-> dirEntryIteratorP	Pointer to the index of the last entry enumerated. For the first iteration, initialize this parameter to the constant <code>vfsIteratorStart</code> . Upon return, this references the next entry in the directory. If <code>infoP</code> is the last entry, this parameter is set to <code>vfsIteratorStop</code> .
<-> infoP	Pointer to the <a href="#">FileInfoType</a> data structure that contains information about the given directory entry. The <code>nameP</code> and <code>nameBufLen</code> fields in this structure must be initialized prior to calling <code>VFSDirEntryEnumerate</code> .

**Result** Returns one of the following error codes:

`errNone` No error.

`expErrEnumerationEmpty`

There are no directory entries left to enumerate.

expErrNotOpen	The file system library necessary for this call has not been installed or has not been opened.
sysErrParamErr	The <code>dirEntryIteratorP</code> is not valid.
vfsErrFileBadRef	The specified file reference is invalid.
vfsErrIsNotADirectory	The specified file reference is valid, but does not point to a directory.
vfsErrNoFileSystem	The VFS Manager cannot find an appropriate file system to handle the request.

**Comments** The directory to be enumerated must first be opened with [VFSFileOpen](#) in order to obtain a file reference. In order to obtain information on all entries in a directory you must make repeated calls to `VFSDirEntryEnumerate` inside a loop. Boundaries on the iteration are the defined constants `vfsIteratorStart` and `vfsIteratorStop`. Before the first call to `VFSDirEntryEnumerate`, `dirEntryIteratorP` should be initialized to `vfsIteratorStart`. Each iteration then changes the value pointed to by `dirEntryIteratorP`. When information on the last entry in the directory is returned, `dirEntryIteratorP` is set to `vfsIteratorStop`.

---

**WARNING!** Creating, renaming, or deleting any file or directory invalidates the enumeration. After any such operation, the enumeration will need to be restarted.

---

**Example** The following code excerpt illustrates how to use `VFSDirEntryEnumerate`.

---

```
FileInfoType info;
FileRef dirRef;
UInt32 dirIterator;
Char *fileName = MemPtrNew(256); // should check for err

// open the directory first, to get the directory reference
// volRefNum must have already been defined
```

## **Virtual File System Manager**

### *VFS Manager Functions*

---

```
err = VFSFileOpen(volRefNum, "/", vfsModeRead, &dirRef);
if(err == errNone) {

    info.nameP = fileName;      // point to local buffer
    info.nameBufLen = 256;
    dirIterator = vfsIteratorStart
    while (dirIterator != vfsIteratorStop) {
        // Get the next file
        err = VFSDirEntryEnumerate (dirRef, &dirIterator,
            &info);
        if (err == errNone) {
            // Do something with the directory entry information
            // Pull the attributes from info.attributes
            // The file name is in fileName
        } else {
            // handle error, possibly by breaking out of the
            loop
        }
    } else {
        // handle directory open error here
    }
    MemPtrFree(fileName);
}
```

---

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

## **VFSExportDatabaseToFile**

**Purpose** Save the specified database to a .pdb or .prc file on an external storage card.

**Declared In** VfsMgr.h

**Prototype** Err VFSExportDatabaseToFile (UInt16 volRefNum,  
const Char \*pathNameP, UInt16 cardNo,  
LocalID dbID)

**Parameters**

-> volRefNum	Volume on which the destination file should be created.
-> pathNameP	Pointer to the complete path and name of the destination file to be created.

-> cardNo Card number on which the .pdb or .prc being exported resides.

-> dbID ID of the database being exported.

**Result** Returns one of the following error codes:

errNone No error

expErrNotEnoughPower

There is insufficient battery power to perform the database export operation.

vfsErrBadName The path name specified in pathNameP is not valid.

**Comments**

This utility function exports a database from main memory to a .pdb or .prc file on an external storage card. This function is the opposite of [VFSImportDatabaseFromFile](#). It first creates the file specified in the pathNameP parameter with [VFSFileCreate](#). After opening the file the Exchange Manager function [ExgDBWrite](#) is called with an internal callback function for exporting the file from the Data Manager. The Exchange Manager makes repeated calls to this callback function, which receives the data back in blocks. Once all the data has been exported, VFS Manager closes the file.

This function is used, for example, to copy applications from main memory to a storage card.

**Compatibility**

Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also**

[VFSExportDatabaseToFileCustom](#), [VFSFileWrite](#),  
[VFSImportDatabaseFromFile](#)

## **VFSExportDatabaseToFileCustom**

**Purpose**

Saves the specified database to a .pdb or .prc file on an external storage card. This function differs from

## **Virtual File System Manager**

### *VFS Manager Functions*

---

[VFSExportDatabaseToFile](#) in that it allows you to track the progress of the export operation.

**Declared In** VfsMgr.h

**Prototype**

```
Err VFSExportDatabaseToFileCustom
    (UInt16 volRefNum, const Char *pathNameP,
     UInt16 cardNo, LocalID dbID,
     VFSExportProcPtr exportProcP, void *userDataP)
```

**Parameters**

-> volRefNum	Volume on which the destination file should be created.
-> pathNameP	Pointer to the complete path and name of the destination file to be created.
-> cardNo	Card number on which the .pdb or .prc being exported resides.
-> dbID	ID of the database being exported.
-> exportProcP	User-defined callback function that tracks the progress of the export. This function should allow the user to cancel the export. Pass NULL if you don't have a progress callback function. See <a href="#">VFSExportProcPtr</a> for the requirements of this function.
-> userDataP	Pointer to any data you want to pass to the callback function specified in <code>exportProcP</code> . This information is not used internally by the VFS Manager. Pass NULL if you don't have a progress callback function or if that function doesn't need any such data.

**Result** Returns one of the following error codes:

`errNone` No error.

`expErrNotEnoughPower` There is insufficient battery power to perform the database export operation.

`vfsErrBadName` The path name specified in `pathNameP` is not valid.

This function can also return any error code other than `errNone` produced by your callback function.

**Comments**

This function is similar to [VFSExportDatabaseToFile](#) in that it exports a database from main memory to a .pdb or .prc file on an external storage card. It extends the functionality by allowing you to specify a callback function that tracks the progress of the export. It first creates the file specified in the `pathNameP` parameter with [VFSFileCreate](#). After opening the file, the Exchange Manager function [ExgDBWrite](#) is called with an internal callback function for exporting the file from the Data Manager. Exchange Manager makes repeated calls to this function, which receives the data back in blocks. The progress tracker, if one has been specified, is also called every time a new chunk of data is passed back. Once all the data has been exported, the VFS Manager closes the file.

This function is used, for example, to copy applications from main memory to a storage card.

**Compatibility**

Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also**

[VFSExportDatabaseToFile](#), [VFSFileWrite](#),  
[VFSImportDatabaseFromFileCustom](#)

## **VFSFileClose**

**Purpose** Closes a file or directory that has been opened with [VFSFileOpen](#).

**Declared In** VfsMgr.h

**Prototype** Err VFSFileClose (FileRef fileRef)

**Parameters** -> fileRef      File reference number returned from [VFSFileOpen](#).

**Result** Returns one of the following error codes:

## Virtual File System Manager

### VFS Manager Functions

---

errNone	No error.
expErrNotOpen	The file system library necessary for this call has not been installed or has not been opened.
vfsErrFileBadRef	The specified file reference is invalid.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

## VFSFileCreate

**Purpose** Create a file. This function cannot be used to create a directory; use [VFSDirCreate](#) instead.

**Declared In** VfsMgr.h

**Prototype** Err VFSFileCreate (UInt16 volRefNum,  
const Char \*pathNameP)

**Parameters** -> volRefNum Reference number of the volume on which to create the file. This volume reference number is returned from [VFSVolumeEnumerate](#).  
-> pathNameP Pointer to the full path of the file to be created. All parts of the path, excluding the filename, must already exist.

**Result** Returns one of the following error codes:

errNone	No error.
expErrNotOpen	The file system library necessary for this call has not been installed or has not been opened.
vfsErrBadName	The pathNameP is invalid.
vfsErrFileAlreadyExists	A file with this name already exists in this location.

`vfsErrNoFileSystem`

The VFS Manager cannot find an appropriate file system to handle the request.

`vfsErrVolumeBadRef`

The volume has not been mounted.

`vfsErrVolumeFull`

There is not enough space left on the volume.

**Comments**

It is the responsibility of the file system library to ensure that all filenames are translated into a format that is compatible with the native format of the file system, such as the 8.3 convention for a FAT file system without long filename support. See [Naming Files](#) in the [Expansion](#) chapter of the *Palm OS Programmer's Companion*, vol. I for a description of how to construct a valid path.

This function does not open the file. Use [VFSFileOpen](#) to open the file.

This function should not be used to create a directory. To create a directory use [VFSDirCreate](#).

**Compatibility**

Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also**

[VFSFileDelete](#)

## **VFSFileDBGetRecord**

**Purpose**

Load a record from an opened .pdb file on an external card into the storage heap.

**Declared In**

`VfsMgr.h`

**Prototype**

```
Err VFSFileDBGetRecord (FileRef ref,  
UInt16 recIndex, MemHandle *recHP,  
UInt8 *recAttrP, UInt32 *uniqueIDP)
```

**Parameters**

`-> ref`

The file reference returned from [VFSFileOpen](#). Note that the open file must be a .pdb file.

## Virtual File System Manager

### VFS Manager Functions

---

-> recIndex	The index of the record to load.
<- recHP	Pointer to the record data's handle in the storage heap. If NULL is returned in this parameter there is either no data in this field or an error occurred reading this data from the file. If the handle is not NULL, you must dispose of the allocated handle using <a href="#">MemHandleFree</a> .
<- recAttrP	Pointer to the attributes of the record. The values returned are identical to the attributes returned from <a href="#">DmRecordInfo</a> . See <a href="#">Record Attribute Constants</a> in the <a href="#">Data and Resource Manager</a> chapter for a description of each attribute. Pass NULL for this parameter if you do not want to retrieve this information.
<- uniqueIDP	Pointer to the unique identifier for this record. Pass NULL for this parameter if you do not want to retrieve this information.

**Result** Returns one of the following error codes:

errNone	No error.
dmErrIndexOutOfRange	The recIndex is out of range.
dmErrNotRecordDB	The file referenced by ref is not a record database.
memErrNotEnoughSpace	There is not enough space in memory for the requested record entry.
sysErrParamErr	A NULL value was passed in for the recHP, recAttrP, and uniqueIDP parameters.
vfsErrBadData	The local offsets (localChunkID) from the top of the .pdb to the start of the raw record data for this entry are out of order.

**Comments** This function is analogous to [DmGetRecord](#) but works with files on an external card rather than databases in main memory. This

function allocates a handle of the appropriate size from the storage heap and returns it in the `recHP` parameter. The caller is responsible for freeing this memory, using [MemHandleFree](#), when it is no longer needed.

---

**NOTE:** This function is not efficient for multiple accesses and should be used sparingly.

---

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSFileReadData](#)

## **VFSFileDBGetResource**

**Purpose** Load a resource into the storage heap from an opened .prc file.

**Declared In** VfsMgr.h

**Prototype** Err VFSFileDBGetResource (FileRef ref,  
DmResType type, DmResID resID, MemHandle \*resHP)

**Parameters**

-> ref	The file reference returned from <a href="#">VFSFileOpen</a> . Note that the open file must be a .prc file.
-> type	The type of resource to load. See the <a href="#">Data and Resource Manager</a> chapter for more information on resources.
-> resID	The ID of resource to load.
<- resHP	Pointer to the resource data handle that was loaded into memory.

**Result** Returns one of the following error codes:

errNone No error.

## **Virtual File System Manager**

### *VFS Manager Functions*

---

`dmErrNotResourceDB`

The file referenced by `ref` is not a resource database.

`dmErrResourceNotFound`

The requested resource was not found.

`memErrNotEnoughSpace`

There is not enough space in memory for the requested resource entries.

`sysErrParamErr` `resHP` is NULL.

#### **Comments**

This function locates the specified resource in the open .prc file. See the *Palm OS File Format Specification* for more information on the layout of .prc files.

Once the resource is found, `VFSFileDBGetResource` allocates a handle of the appropriate size in the storage heap and reads it into memory. The handle to this memory location is returned through the `resHP` parameter. The caller is responsible for freeing this memory, using [MemHandleFree](#), when it is no longer needed.

---

**NOTE:** This function is not efficient for multiple accesses and should be used sparingly.

---

#### **Compatibility**

Implemented only if the [VFS Manager Feature Set](#) is present.

## VFSFileDBInfo

**Purpose** Get information about a database represented by an open .prc or .pdb file.

**Declared In** VfsMgr.h

**Prototype**

```
Err VFSFileDBInfo (FileRef ref, Char *nameP,
UInt16 *attributesP, UInt16 *versionP,
UInt32 *crDateP, UInt32 *modDateP,
UInt32 *bckUpDateP, UInt32 *modNumP,
MemHandle *appInfoHP, MemHandle *sortInfoHP,
UInt32 *typeP, UInt32 *creatorP,
UInt16 *numRecordsP)
```

<b>Parameters</b>	-> ref	The file reference returned from <a href="#">VFSFileOpen</a> . Note that the open file must be a .prc or .pdb file.
	<- nameP	Pointer to a 32-byte character array in which the database name is returned. Pass NULL for this parameter if you do not want to retrieve the database name.
	<- attributesP	Pointer to the database attributes stored in the file. The values returned are identical to the attributes returned from <a href="#">DmDatabaseInfo</a> . See the <a href="#">Database Attribute Constants</a> section for a description of each attribute. Pass NULL for this parameter if you do not want to retrieve the database's attributes.
	<- versionP	Pointer to the application-specific version number of the database. The default version number is zero (0). Pass NULL for this parameter if you do not want to retrieve the version number.
	<- crDateP	Pointer to the date the database was created, expressed in seconds since midnight (00:00:00) January 1, 1904. Pass NULL for this parameter if you do not want to retrieve the creation date.

## **Virtual File System Manager**

### *VFS Manager Functions*

---

<- modDateP	Pointer to the date the database was last modified, expressed in seconds since midnight (00:00:00) January 1, 1904. A database's modification date is updated only if a change has been made to the database when it is opened with write access. Pass NULL for this parameter if you do not want to retrieve the database's modification date.
<- bckUpDateP	Pointer to the date the database was last backed up, expressed in seconds since midnight (00:00:00) January 1, 1904. Pass NULL for this parameter if you do not want to retrieve the database's backup date.
<- modNumP	Pointer to the number of times the database was modified. This number is updated every time a record is added, modified, or deleted. Pass NULL for this parameter if you do not want to retrieve the modification count.
<- appInfoHP	Pointer to the application info block handle. If NULL is returned in this parameter, either there is no data in this field or an error occurred reading this data from the file. If a value other than NULL is returned, you must dispose of the allocated handle using <a href="#">MemHandleFree</a> . If you do not want to retrieve the application info block, pass NULL for this parameter
<- sortInfoHP	Pointer to the sort info block handle. If NULL is returned in this parameter, either there is no data in this field or an error occurred reading this data from the file. If a value other than NULL is returned, you must dispose of the allocated handle using <a href="#">MemHandleFree</a> . Pass NULL for this parameter if you do not want to retrieve the sort info block handle.

<- typeP

Pointer to the type of database as it was created. This may be a user-defined database type or a database type defined by the Palm OS. Some of the more common database types returned here are:

Type	Description
'appl'	Standard Palm™ application (resource database)
'libr'	Standard shared library
'libf'	File system shared library
'libs'	Slot driver shared library
'data'	Standard Palm data file (record database)

<- creatorP

Pass NULL for this parameter if you do not want to retrieve the database's type.

<- numRecordsP

Pointer to the database's creator. Pass NULL for this parameter if you do not want to retrieve this information.

<- numRecordsP

Pointer to the number of records in the database. Pass NULL for this parameter if you do not want to retrieve this information.

**Result** Returns one of the following error codes:

errNone No error

memErrNotEnoughSpace

There is not enough space in memory for the database header.

vfsErrBadData

The file referenced by the ref parameter is too small to contain a database header, or the database header is corrupted.

## **Virtual File System Manager**

### *VFS Manager Functions*

---

- Comments** This function is analogous to [DmDatabaseInfo](#), but it works with files on an external card rather than with databases in main memory. See the *Palm OS File Format Specification* for a description of the header block in .prc and .pdb files.
- Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.
- See Also** [VFSFileGetAttributes](#), [VFSFileGetDate](#)

## **VFSFileDelete**

- Purpose** Deletes a closed file or directory.
- Declared In** VfsMgr.h
- Prototype** Err VFSFileDelete(UInt16 volRefNum,  
const Char \*pathNameP)
- Parameters**
- |              |   |
|--------------|---|
| -> volRefNum | Volume reference number returned from<br><a href="#">VFSVolumeEnumerate</a> . |
| -> pathNameP | Pointer to the full path of the file or directory to<br>be deleted.           |
- Result** Returns one of the following error codes:
- |                             |   |
|-----------------------------|---|
| errNone                     | No error.   |
| expErrNotOpen               | The file system library necessary for this call<br>has not been installed or has not been opened. |
| vfsErrBadName               | The path name specified in pathNameP is not<br>valid.   |
| vfsErrDirNotEmpty           | The directory being deleted is not empty.   |
| vfsErrFileStillOpen         | The file is still open.   |
| vfsErrFileNotFoundException | The file could not be found.  |

<code>vfsErrFilePermissionDenied</code>	The requested permissions could not be granted.
<code>vfsErrNoFileSystem</code>	The VFS Manager cannot find an appropriate file system to handle the request.
<code>vfsErrVolumeBadRef</code>	The volume has not been mounted.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

## **VFSFileEOF**

**Purpose** Get end-of-file status for an open file. This function only operates on files and cannot be used with directories.

**Declared In** `VfsMgr.h`

**Prototype** `Err VFSFileEOF (FileRef fileRef)`

**Parameters** `- > fileRef` File reference returned from [VFSFileOpen](#).

**Result** Returns one of the following error codes:

`errNone` No error. File pointer is not at end of the file.

`vfsErrFileEOF` The file pointer is at the end of file.

`expErrNotOpen` The file system library necessary for this call has not been installed or has not been opened.

`vfsErrFileBadRef`

The specified file reference is invalid.

`vfsErrIsADirectory`

The specified file reference points to a directory instead of a file. This is an invalid operation on a directory.

## Virtual File System Manager

### VFS Manager Functions

---

`vfsErrNoFileSystem`

The VFS Manager cannot find an appropriate file system to handle the request.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

## VFSFileGetAttributes

**Purpose** Obtain the attributes of an open file or directory.

**Declared In** `VfsMgr.h`

**Prototype** `Err VFSFileGetAttributes (FileRef fileRef,  
                  UInt32 *attributesP)`

**Parameters** `-> fileRef` File reference returned from [VFSFileOpen](#).  
`<- attributesP` Pointer to the attributes associated with the file or directory. See [File and Directory Attributes](#) for a list of values that can be returned through this parameter.

**Result** Returns one of the following error codes:

`errNone` No error

`expErrNotOpen` The file system library necessary for this call has not been installed or has not been opened.

`vfsErrFileBadRef`  
The specified file reference is invalid.

`vfsErrNoFileSystem`  
The VFS Manager cannot find an appropriate file system to handle the request.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSFileInfo](#), [VFSFileGetDate](#), [VFSFileSetAttributes](#)

## VFSFileGetDate

**Purpose** Obtain the dates on an open file or directory.

**Declared In** VfsMgr.h

**Prototype** Err VFSFileGetDate (FileRef fileRef,  
UInt16 whichDate, UInt32 \*dateP)

**Parameters** -> fileRef File reference returned from [VFSFileOpen](#).

-> whichDate Specifies which date—creation, modification, or last access—you want. Supply one of the following values:

vfsFileDateCreated

vfsFileDateModified

vfsFileDateAccessed

Note that not all file systems are required to support the above dates. If the supplied date type is not supported by the file system, `VFSFileGetDate` returns `expErrUnsupportedOperation`.

<- dateP Pointer to the requested date. This field is expressed in the standard Palm OS date format—the number of seconds since midnight (00:00:00) January 1, 1904.

**Result** Returns one of the following error codes:

errNone No error.

expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.

expErrUnsupportedOperation

The specified date type is not supported by the underlying file system.

## Virtual File System Manager

### VFS Manager Functions

---

`vfsErrFileBadRef`

The specified file reference is invalid.

`sysErrParamErr` The `whichDate` parameter is not one of the defined constants.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSFileInfo](#), [VFSFileGetAttributes](#), [VFSFileSetDate](#)

## VFSFileOpen

**Purpose** Opens a file or directory and returns a reference for it.

**Declared In** `VfsMgr.h`

**Prototype** `Err VFSFileOpen (UInt16 volRefNum,  
const Char *pathNameP, UInt16 openMode,  
FileRef *fileRefP)`

**Parameters** `-> volRefNum` The volume reference number returned from [VFSVolumeEnumerate](#).

`-> pathNameP` Pointer to the full path of the file or directory to be opened. This must be a valid path. It cannot be empty and can not contain null characters. The format of the pathname should match what the underlying file system supports. See “[Naming Files](#)” in [Chapter 7](#), “[Expansion](#),” of the *Palm OS Programmer’s Companion*, vol. I for a description of how to construct a valid path.

`-> openMode` Mode to use when opening the file. See the [Open Mode Constants](#) section for a list of accepted modes.

<- fileRefP      Pointer to the opened file or directory reference which is supplied to various other VFSFile... operations. This value is filled in on return.

**Result**      Returns one of the following error codes:

errNone      No error.

expErrCardReadOnly      The open mode requested includes write access but the file is read-only.

expErrNotOpen      The file system library necessary for this call has not been installed or has not been opened.

vfsErrBadName      The pathNameP parameter is invalid.

vfsErrFileNotFoundException      The specified file or directory could not be found.

vfsErrFilePermissionDenied      The file cannot be opened in the requested open mode, or it has already been opened with vfsModeExclusive.

vfsErrVolumeBadRef      The specified volume has not been mounted.

**Compatibility**      Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also**      [VFSFileClose](#), [VFSDirEntryEnumerate](#)

## VFSFileRead

**Purpose** Read data from a file into the dynamic heap. This function only operates on files and cannot be used with directories; use [VFSDirEntryEnumerate](#) to explore the contents of a directory.

**Declared In** VfsMgr.h

**Prototype** Err VFSFileRead (FileRef fileRef,  
UInt32 numBytes, void \*bufP,  
UInt32 \*numBytesReadP)

**Parameters**

-> fileRef	File reference returned from <a href="#">VFSFileOpen</a> .
-> numBytes	Number of bytes to read.
<- bufP	Pointer to the destination chunk where the data is to be stored. This can be a pointer to any writable memory.
<- numBytesReadP	Pointer to an unsigned integer that reflects the number of bytes actually read. This value is set on return and does not need to be initialized. If no bytes are read the value is set to zero. Pass NULL for this parameter if you do not need to know how many bytes were read.

**Result** Returns one of the following error codes:

errNone	No error.
expErrNotOpen	The file system library necessary for this call has not been installed or has not been opened.
vfsErrFileBadRef	The specified file reference is invalid.
vfsErrFileEOF	The end of the file has been reached.
vfsErrFilePermissionDenied	Read permission is not enabled for this file.

`vfsErrIsADirectory`

The specified file reference is for a directory instead of a file. This is an invalid operation on a directory.

`vfsErrNoFileSystem`

The VFS Manager cannot find an appropriate file system to handle the request.

**Comments** The file system does not use [DmWrite](#) and cannot be used to read data into the storage heap.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSFileReadData](#), [VFSFileWrite](#),  
[VFSImportDatabaseFromFile](#)

## VFSFileReadData

**Purpose** Read data from a file into a chunk of memory in the storage heap. This function only operates on files and cannot be used with directories; use [VFSDirEntryEnumerate](#) to explore the contents of a directory.

**Declared In** VfsMgr.h

**Prototype** Err VFSFileReadData (FileRef fileRef,  
                  UInt32 numBytes, void \*bufBaseP, UInt32 offset,  
                  UInt32 \*numBytesReadP)

**Parameters**

-> fileRef	File reference returned in <a href="#">VFSFileOpen</a> .
-> numBytes	Number of bytes to read.
<- bufBaseP	Pointer to the destination chunk in the storage heap where the data is to be stored. This pointer must be obtained through the appropriate call to the <a href="#">Memory Manager</a> API.
-> offset	Offset, in bytes, within the bufBaseP chunk where the data is to be written.

## Virtual File System Manager

### VFS Manager Functions

---

<- numBytesReadP

Pointer to an unsigned integer that reflects the number of bytes actually read. This value is set on return and does not need to be initialized. If no bytes are read, the value is set to zero. Pass NULL for this parameter if you do not need to know how many bytes were read.

**Result** Returns one of the following error codes:

errNone No error.

expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.

vfsErrFileBadRef  
The specified file reference is invalid.

vfsErrFileEOF The end of the file has been reached.

vfsErrFilePermissionDenied  
Read permission is not enabled for this file.

vfsErrIsADirectory  
The specified file reference is for a directory instead of a file. This is an invalid operation on a directory.

vfsErrNoFileSystem  
The VFS Manager cannot find an appropriate file system to handle the request.

**Comments** When data is read from an external card with `VFSFileReadData`, it is copied into a chunk of memory in the storage heap. This chunk **must** be allocated by the application before the call to `VFSFileReadData`. This function calls `DmWrite` to put the data in the storage heap.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSFileRead](#), [VFSFileWrite](#)

## VFSFileRename

<b>Purpose</b>	Rename a closed file or directory. This function cannot be used to move a file to another directory within the file system.												
<b>Declared In</b>	VfsMgr.h												
<b>Prototype</b>	Err VFSFileRename (UInt16 volRefNum, const Char *pathNameP, const Char *newNameP)												
<b>Parameters</b>	<table><tr><td>- &gt; volRefNum</td><td>Volume reference number returned from <a href="#">VFSVolumeEnumerate</a>.</td></tr><tr><td>- &gt; pathNameP</td><td>Pointer to the full path of the file or directory to be renamed.</td></tr><tr><td>- &gt; newNameP</td><td>Pointer to the new filename. Note that this is the name of the file only and does not include the path to the file.</td></tr></table>	- > volRefNum	Volume reference number returned from <a href="#">VFSVolumeEnumerate</a> .	- > pathNameP	Pointer to the full path of the file or directory to be renamed.	- > newNameP	Pointer to the new filename. Note that this is the name of the file only and does not include the path to the file.						
- > volRefNum	Volume reference number returned from <a href="#">VFSVolumeEnumerate</a> .												
- > pathNameP	Pointer to the full path of the file or directory to be renamed.												
- > newNameP	Pointer to the new filename. Note that this is the name of the file only and does not include the path to the file.												
<b>Result</b>	Returns one of the following error codes:  <table><tr><td>errNone</td><td>No error.</td></tr><tr><td>expErrNotOpen</td><td>The file system library necessary for this call has not been installed or has not been opened.</td></tr><tr><td>vfsErrBadName</td><td>The name provided in either pathNameP or newNameP is invalid. This is also returned if the string pointed to by newNameP is a path, rather than a filename.</td></tr><tr><td>vfsErrFileAlreadyExists</td><td>A file with the new name already exists in this location.</td></tr><tr><td>vfsErrFileNotFoundException</td><td>The source file could not be found.</td></tr><tr><td>vfsErrFilePermissionDenied</td><td>Write permission is not enabled for this file.</td></tr></table>	errNone	No error.	expErrNotOpen	The file system library necessary for this call has not been installed or has not been opened.	vfsErrBadName	The name provided in either pathNameP or newNameP is invalid. This is also returned if the string pointed to by newNameP is a path, rather than a filename.	vfsErrFileAlreadyExists	A file with the new name already exists in this location.	vfsErrFileNotFoundException	The source file could not be found.	vfsErrFilePermissionDenied	Write permission is not enabled for this file.
errNone	No error.												
expErrNotOpen	The file system library necessary for this call has not been installed or has not been opened.												
vfsErrBadName	The name provided in either pathNameP or newNameP is invalid. This is also returned if the string pointed to by newNameP is a path, rather than a filename.												
vfsErrFileAlreadyExists	A file with the new name already exists in this location.												
vfsErrFileNotFoundException	The source file could not be found.												
vfsErrFilePermissionDenied	Write permission is not enabled for this file.												

## **Virtual File System Manager**

### *VFS Manager Functions*

---

`vfsErrNoFileSystem`

The VFS Manager cannot find an appropriate file system to handle the request.

`vfsErrVolumeBadRef`

The volume has not been mounted.

`vfsErrVolumeFull`

There is not enough space left on the volume.

---

#### **Comments**

**WARNING!** This function invalidates directory enumeration. You cannot continue enumerating files after renaming one of them with this function. If you need to operate on additional files in the directory, you must first restart the enumeration.

---

#### **Example**

Below is an example of how to use `VFSFileRename`. Note that the renamed file remains in the `/PALM/Programs` directory; `VFSFileRename` can't be used to move files from one directory to another.

```
// volRefNum must have been previously defined; most likely,  
// it was returned by VFSVolumeEnumerate  
  
err = VFSFileRename(volRefNum, "/PALM/Programs/foo.prc",  
    "bar.prc");  
if (err != errNone) {  
    // handle error...  
}
```

---

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

## VFSFileResize

**Purpose** Change the size of an open file. This function only operates on files and cannot be used with directories.

**Declared In** VfsMgr.h

**Prototype** Err VFSFileResize (FileRef fileRef,  
 UInt32 newSize)

**Parameters**

-> fileRef	File reference returned from <a href="#">VFSFileOpen</a> .
-> newSize	The desired new size of the file. This can be larger or smaller than the current file size.

**Result** Returns one of the following error codes:

errNone No error.

expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.

vfsErrFileBadRef

The specified file reference is invalid.

vfsErrIsADirectory

The specified file reference points to a directory instead of a file. This is an invalid operation on a directory.

## **Virtual File System Manager**

### *VFS Manager Functions*

---

`vfsErrNoFileSystem`

The VFS Manager cannot find an appropriate file system to handle the request.

`vfsErrVolumeFull`

There is not enough space left on the volume.

**Comments** The location of the file pointer is undefined after a call to this function.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSFileSize](#)

## **VFSFileSeek**

**Purpose** Set the position within an open file from which to read or write. This function only operates on files and cannot be used with directories.

**Declared In** `VfsMgr.h`

**Prototype** `Err VFSFileSeek (FileRef fileRef,  
FileOrigin origin, Int32 offset)`

**Parameters**

-> <code>fileRef</code>	File reference returned from <a href="#">VFSFileOpen</a> .
-> <code>origin</code>	Origin to use when calculating the new position. The <code>offset</code> parameter indicates the desired new position relative to this origin, which can be one of the following:  <code>vfsOriginBeginning</code> The beginning of the file.  <code>vfsOriginCurrent</code> The current position within the file.  <code>vfsOriginEnd</code> The end of the file. Only negative offsets are allowed when <code>origin</code> is set to <code>vfsOriginEnd</code> .

-> offset      Offset, either positive or negative, from the origin to which the current position should be set. A value of zero (0) positions you at the specified origin.

**Result**      Returns one of the following error codes:

errNone      No error.

expErrNotOpen      The file system library necessary for this call has not been installed or has not been opened.

vfsErrFileBadRef      The specified file reference is invalid.

vfsErrFileEOF      The file pointer is at the end of file.

vfsErrIsADirectory      The specified file reference points to a directory instead of a file. This is an invalid operation on a directory.

sysErrParamErr      The specified origin is not one of the defined constants.

**Comments**      During a call to this function, if the resulting position would be beyond the end of the file, it sets the position to the end of the file.

**Compatibility**      Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also**      [VFSFileSize](#), [VFSFileTell](#)

## VFSFileSetAttributes

**Purpose**      Change the attributes of an open file or directory.

**Declared In**      VfsMgr.h

**Prototype**      Err VFSFileSetAttributes (FileRef fileRef,  
                  UInt32 attributes)

## Virtual File System Manager

### VFS Manager Functions

---

<b>Parameters</b>	-> fileRef	File reference returned from <a href="#">VFSFileOpen</a> .
	-> attributes	Attributes to associate with the file or directory. See <a href="#">File and Directory Attributes</a> for a list of values you can use when setting this parameter:
<b>Result</b>	Returns one of the following error codes:	
	errNone	No error.
	expErrNotOpen	The file system library necessary for this call has not been installed or has not been opened.
	sysErrParamErr	One of the parameters is invalid.
	vfsErrFileBadRef	The specified file reference is invalid.
	vfsErrNoFileSystem	The VFS Manager cannot find an appropriate file system to handle the request.
<b>Comments</b>	<b>NOTE:</b> You cannot use this function to set the <code>vfsFileAttrDirectory</code> or <code>vfsFileAttrVolumeLabel</code> attributes. The <code>vfsFileAttrDirectory</code> is set when you call <a href="#">VFSDirCreate</a> . The <code>vfsFileAttrVolumeLabel</code> is set when you call <a href="#">VFSVolumeSetLabel</a> . This function may fail when setting other attributes, depending on the underlying file system.	
<b>Compatibility</b>	Implemented only if the <a href="#">VFS Manager Feature Set</a> is present.	
<b>See Also</b>	<a href="#">VFSFileGetAttributes</a> , <a href="#">VFSFileSetDate</a>	

## VFSFileSetDate

**Purpose** Changes the dates on an open file or directory.

**Declared In** VfsMgr.h

**Prototype** Err VFSFileSetDate (FileRef fileRef,  
UInt16 whichDate, UInt32 date)

**Parameters** -> fileRef File reference returned in [VFSFileOpen](#).

-> whichDate Specifies which date—creation, modification, or last access—to modify. Supply one of the following values:

vfsFileDateCreated

vfsFileDateModified

vfsFileDateAccessed

Note that not all file systems are required to support the above dates. If the supplied date type is not supported by the file system, `VFSFileGetDate` returns `expErrUnsupportedOperation`.

-> date The new date. This field should be expressed in the standard Palm OS date format — the number of seconds since midnight (00:00:00) January 1, 1904.

**Result** Returns one of the following error codes:

errNone No error.

expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.

expErrUnsupportedOperation

The specified date type is not supported by the underlying file system.

sysErrParamErr The `whichDate` parameter is not one of the defined constants.

## **Virtual File System Manager**

### *VFS Manager Functions*

---

`vfsErrFileBadRef`

The specified file reference is invalid.

`vfsErrFilePermissionDenied`

Write permission is not enabled for this file.

`vfsErrNoFileSystem`

The VFS Manager cannot find an appropriate file system to handle the request.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSFileGetDate](#), [VFSFileSetAttributes](#)

## **VFSFileSize**

**Purpose** Obtain the size of an open file. This function only operates on files and cannot be used with directories.

**Declared In** `VfsMgr.h`

**Prototype** `Err VFSFileSize (FileRef fileRef,  
UInt32 *fileSizeP)`

**Parameters** `-> fileRef` File reference returned from [VFSFileOpen](#).  
`<- fileSizeP` Pointer to the size of the open file.

**Result** Returns one of the following error codes:

`errNone` No error.

`expErrNotOpen` The file system library necessary for this call has not been installed or has not been opened.

`vfsErrFileBadRef`

The specified file reference is invalid.

`vfsErrIsADirectory`

The specified file reference points to a directory instead of a file. This is an invalid operation on a directory.

`vfsErrNoFileSystem`

The VFS Manager cannot find an appropriate file system to handle the request.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSFileResize](#), [VFSFileTell](#), [VFSVolumeSize](#)

## **VFSFileTell**

**Purpose** Get the current position of the file pointer within an open file. This function only operates on files and cannot be used with directories.

**Declared In** `VfsMgr.h`

**Prototype** `Err VFSFileTell (FileRef fileRef,  
                  UInt32 *filePosP)`

**Parameters** `-> fileRef` File reference returned from [VFSFileOpen](#).  
`<- filePosP` Pointer to the current file position.

**Result** Returns one of the following error codes:

`errNone` No error.

`expErrNotOpen` The file system library necessary for this call has not been installed or has not been opened.

`vfsErrFileBadRef`

The specified file reference is invalid.

`vfsErrIsADirectory`

The specified file reference points to a directory instead of a file. This is an invalid operation on a directory.

`vfsErrNoFileSystem`

The VFS Manager cannot find an appropriate file system to handle the request.

## **Virtual File System Manager**

### *VFS Manager Functions*

---

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSFileSeek](#), [VFSFileSize](#)

## **VFSFileWrite**

**Purpose** Write data to an open file. This function only operates on files and cannot be used with directories.

**Declared In** VfsMgr.h

**Prototype** Err VFSFileWrite (FileRef fileRef,  
                  UInt32 numBytes, const void \*dataP,  
                  UInt32 \*numBytesWrittenP)

**Parameters**

-> fileRef	File reference returned from <a href="#">VFSFileOpen</a> .
-> numBytes	The number of bytes to write.
-> dataP	Pointer to the data that is to be written.
<- numBytesWrittenP	Pointer to an unsigned integer that reflects the number of bytes actually written. This value is set on return and does not need to be initialized. If no bytes are written the value is set to zero. Pass NULL for this parameter if you do not need to know how many bytes were written.

**Result** Returns one of the following error codes:

errNone	No error.
expErrNotOpen	The file system library necessary for this call has not been installed or has not been opened.
vfsErrFileBadRef	The specified file reference is invalid.
vfsErrFilePermissionDenied	Write permission is not enabled for this file.

`vfsErrIsADirectory`

The specified file reference points to a directory instead of a file. This is an invalid operation on a directory.

`vfsErrNoFileSystem`

The VFS Manager cannot find an appropriate file system to handle the request.

`vfsErrVolumeFull`

There is not enough space left on the volume.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also**

[VFSExportDatabaseToFile](#),  
[VFSExportDatabaseToFileCustom](#), [VFSFileRead](#),  
[VFSFileReadData](#)

## VFSGetDefaultDirectory

**Purpose** Determine the default location on the given volume for files of a particular type.

**Declared In** `VfsMgr.h`

**Prototype** `Err VFSGetDefaultDirectory (UInt16 volRefNum,  
const Char *fileTypeStr, Char *pathStr,  
UInt16 *bufLenP)`

**Parameters** `-> volRefNum` Volume reference number returned from [VFSVolumeEnumerate](#).

`-> fileTypeStr` Pointer to the requested file type, as a null-terminated string. The file type may either be a MIME media type/subtype pair, such as "image/jpeg", "text/plain", or "audio/basic"; or a file extension, such as ".jpeg."

`<- pathStr` Pointer to the buffer which receives the default directory path for the requested file type.

## **Virtual File System Manager**

### *VFS Manager Functions*

---

<-> bufLenP      Pointer to the size of the path. Set this to the size of pathStr buffer on input. Reflects the number of bytes copied to pathStr on output.

**Result**      Returns one of the following error codes:

errNone      No error.

vfsErrBadName      There is no default directory registered for the requested file type.

vfsErrBufferOverflow

A match was found, but the pathStr buffer is too small to hold the resulting path string. A partial path is returned in pathStr.

vfsErrFileNotFoundException

No match was found for the specified volume. The error could have occurred with either the media type specified for this volume or the file type requested.

**Comments**

This function returns the complete path to the default directory registered for the specified file type. A default directory can be registered for each type of media supported. The directory should be registered under media and file type. Note that this directory is typically a “root” directory for the file type; any subdirectories under this root directory should also be searched for files of the appropriate type.

This function can be used by an image viewer application, for example, to find the directory containing images without having to know what type of media the volume was on. This could be “/DCIM”, “/images”, or something else depending on the type of media.

**Compatibility**

Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also**

[VFSDirEntryEnumerate](#), [VFSRegisterDefaultDirectory](#),  
[VFSUnregisterDefaultDirectory](#)

## **VFSImportDatabaseFromFile**

<b>Purpose</b>	Creates a database from a .pdb or .prc file on an external storage card.								
<b>Declared In</b>	VfsMgr.h								
<b>Prototype</b>	Err VFSImportDatabaseFromFile (UInt16 volRefNum, const Char *pathNameP, UInt16 *cardNoP, LocalID *dbIDP)								
<b>Parameters</b>	<table><tr><td>-&gt; volRefNum</td><td>Volume on which the source file resides.</td></tr><tr><td>-&gt; pathNameP</td><td>Pointer to the full path and name of the source file.</td></tr><tr><td>&lt;- cardNoP</td><td>Pointer to a variable that receives the card number of the newly-created database. If the database already resides in the storage heap, the card number of the existing database is returned along with the error dmErrAlreadyExists.</td></tr><tr><td>&lt;- dbIDP</td><td>Pointer to a variable that receives the database ID of the new database. If the database already resides in the storage heap, the database ID of the existing database is returned along with the error dmErrAlreadyExists.</td></tr></table>	-> volRefNum	Volume on which the source file resides.	-> pathNameP	Pointer to the full path and name of the source file.	<- cardNoP	Pointer to a variable that receives the card number of the newly-created database. If the database already resides in the storage heap, the card number of the existing database is returned along with the error dmErrAlreadyExists.	<- dbIDP	Pointer to a variable that receives the database ID of the new database. If the database already resides in the storage heap, the database ID of the existing database is returned along with the error dmErrAlreadyExists.
-> volRefNum	Volume on which the source file resides.								
-> pathNameP	Pointer to the full path and name of the source file.								
<- cardNoP	Pointer to a variable that receives the card number of the newly-created database. If the database already resides in the storage heap, the card number of the existing database is returned along with the error dmErrAlreadyExists.								
<- dbIDP	Pointer to a variable that receives the database ID of the new database. If the database already resides in the storage heap, the database ID of the existing database is returned along with the error dmErrAlreadyExists.								
<b>Result</b>	Returns one of the following error codes:  <table><tr><td>errNone</td><td>No error.</td></tr><tr><td>dmErrAlreadyExists</td><td>The .prc or .pdb file already exists in the storage heap. In this case the cardNoP and dbIDP are set to point to the existing file.</td></tr><tr><td>expErrNotEnoughPower</td><td>There is insufficient battery power to complete the requested operation.</td></tr></table>	errNone	No error.	dmErrAlreadyExists	The .prc or .pdb file already exists in the storage heap. In this case the cardNoP and dbIDP are set to point to the existing file.	expErrNotEnoughPower	There is insufficient battery power to complete the requested operation.		
errNone	No error.								
dmErrAlreadyExists	The .prc or .pdb file already exists in the storage heap. In this case the cardNoP and dbIDP are set to point to the existing file.								
expErrNotEnoughPower	There is insufficient battery power to complete the requested operation.								

## **Virtual File System Manager**

### *VFS Manager Functions*

---

`vfsErrBadName` The path name specified in `pathNameP` is not valid.

<b>Comments</b>	This utility function imports a .pdb or .prc file resident on an external storage card into a new database in the storage heap. It first calls <a href="#">VFSFileOpen</a> to open the file specified in <code>pathNameP</code> . Assuming that a corresponding .prc or .pdb does not already exist in the storage heap, <a href="#">VFSImportDatabaseFromFile</a> calls the Exchange Manager function <a href="#">ExgDBRead</a> with an internal callback function for importing a file to the Data Manager. The Exchange Manager makes repeated calls to this function, which passes the data back in blocks. Once the file has been successfully imported, the owner (the imported file, if it's an executable, or the associated application if it is not) is sent a <a href="#">sysAppLaunchCmdSyncNotify</a> launch code to make it aware of the new database.  This function doesn't provide any progress indication to the user. If you need to provide feedback to the user as the file import progresses, use <a href="#">VFSImportDatabaseFromFileCustom</a> instead.  This function is used, for example, to copy applications from a storage card to main memory.
<b>Compatibility</b>	Implemented only if the <a href="#">VFS Manager Feature Set</a> is present.
<b>See Also</b>	<a href="#">VFSExportDatabaseToFile</a> , <a href="#">VFSFileRead</a>

## **VFSImportDatabaseFromFileCustom**

<b>Purpose</b>	Create a database from the specified .pdb or .prc file on an external storage card. This function differs from
----------------	--

[VFSImportDatabaseFromFile](#) in that it allows you to track the progress of the import operation.

**Declared In** VfsMgr.h

**Prototype**

```
Err VFSImportDatabaseFromFileCustom  
(UInt16 volRefNum, const Char *pathNameP,  
UInt16 *cardNoP, LocalID *dbIDP,  
VFSImportProcPtr importProcP, void *userDataP)
```

<b>Parameters</b>	-> volRefNum	Volume on which the source file resides.
	-> pathNameP	Pointer to the full path and name of the source file.
	<- cardNoP	Pointer to the variable that receives the card number of the newly-created database. If the database already resides in the storage heap, the card number of the existing database is returned along with the error dmErrAlreadyExists.
	<- dbIDP	Pointer to the variable that receives the database ID of the new database. If the database already resides in the storage heap, the database ID of the existing database is returned along with the error dmErrAlreadyExists.
	-> importProcP	User-defined callback function that tracks the progress of the import. This function should allow the user to cancel the import. Pass NULL if you don't have a progress callback function. See <a href="#">VFSImportProcPtr</a> for the requirements of this function.

## Virtual File System Manager

### VFS Manager Functions

---

-> userDataP	Pointer to any data you want to pass to the callback function specified in importProcP. This information is not used internally by the VFS Manager. Pass NULL if you don't have a progress callback function, or if that function doesn't need any such data.
<b>Result</b>	Returns one of the following error codes:
errNone	No error
vfsErrBadName	The path name specified in pathNameP is not valid.
expErrNotEnoughPower	The power required to import a database is not available.
dmErrAlreadyExists	The .prc or .pdb file already exists in main memory. In this case the cardNoP and dbIDP are set to point to the existing file.
<b>Comments</b>	This function is similar to <a href="#">VFSImportDatabaseFromFile</a> in that it imports a .pdb or .prc file on an external storage card into a new database on the storage heap. It extends the functionality by allowing you to specify a callback function that tracks the progress of the export. It first calls <a href="#">VFSFileOpen</a> to open the file specified in pathNameP. If a corresponding .prc or .pdb does not already exist in main memory, it calls the Exchange Manager function <a href="#">ExgDBRead</a> with an internal callback function for importing the file from the Data Manager. The Exchange Manager makes repeated calls to this function, which receives the data back in blocks. The progress tracker, if one has been specified, is also called every time a new chunk of data is passed back. Once the file has been successfully imported, the owner (the imported file, if it's an executable, or the associated application if it is not) is sent a <a href="#">sysAppLaunchCmdSyncNotify</a> launch code to make it aware of the new database.  This function is used, for example, to copy applications from a storage card to main memory.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSFileRead](#), [VFSExportDatabaseToFileCustom](#)

## VFSInstallFSLib

**Purpose** Install a file system library so that the VFS Manager can use it.

**Declared In** VfsMgr.h

**Prototype** Err VFSInstallFSLib (UInt32 creator,  
                  UInt16 \*fsLibRefNumP)

**Parameters** -> creator Creator ID of the database containing the file system library to be installed.

<- fsLibRefNumP  
Pointer to the reference number for the newly installed file system library. Supply NULL for this parameter if you don't need the library reference number.

**Result** If the file system library was loaded and installed without error, errNone is returned. Any error generated by the underlying file system while opening the file system library or determining its type will be returned from VFSInstallFSLib. Other errors that can be generated during the file system library installation process include:

`expErrIncompatibleAPIVer`

The file system library has an incompatible API version.

`expErrNotOpen`

The file system library necessary for this call has not been installed or has not been opened.

`memErrNotEnoughSpace`, `memErrChunkNotLocked`, or

`memErrChunkLocked`

A memory problem occurred while inserting the library reference into the list of installed libraries.

## **Virtual File System Manager**

### *VFS Manager Functions*

---

sysErrLibNotFound, sysErrNoFreeRAM,  
sysErrNoFreeLibSlots (or some other  
error returned from the library's install entry  
point)  
An error occurred while loading the library.

**Comments** This function calls [SysLibLoad](#) to load the file system library into the library table. Once loaded the appropriate file system is asked to open the library. At boot time VFSInstallFSLib is called internally by the Expansion Manager to load all installed file system libraries and initialize them for use.

VFSInstallFSLib is not normally called by applications.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSRemoveFSLib](#)

## **VFSRegisterDefaultDirectory**

**Purpose** Registers a specific directory as the default location for files of a given type on a particular kind of external storage card. This function is generally called by a slot driver for files and media types that are supported by that slot driver.

**Declared In** VfsMgr.h

**Prototype** Err VFSRegisterDefaultDirectory  
(const Char \*fileTypeStr, UInt32 mediaType,  
const Char \*pathStr)

**Parameters** -> fileTypeStr Pointer to the file type to register. This is a null-terminated string that can either be a MIME media type/subtype pair, such as "image/jpeg", "text/plain", or "audio/basic"; or a file extension, such as ".jpeg".

-> mediaType	Type of card media for which the default directory is being associated. See <a href="#">Defined Media Types</a> in the <a href="#">Expansion Manager</a> chapter for the list of accepted values.
-> pathStr	Pointer to the default directory path to be associated with the specified file type. This string must be null-terminated, and must be the full path to the directory.

<b>Result</b>	Returns one of the following error codes:
errNone	No error.
sysErrParamErr	Either the <code>fileTypeStr</code> parameter is NULL or the <code>pathStr</code> parameter is NULL.
<code>vfsErrFileAlreadyExists</code>	A default directory has already been registered for this file type on the specified card media type.

## **Virtual File System Manager**

### *VFS Manager Functions*

---

<b>Comments</b>	This function first verifies that a default directory has not already been registered for the specified combination of file type and media type, and returns <code>vfsErrFileAlreadyExists</code> if one has been registered. To change an existing entry in the registry, you must first remove the existing entry with a call to <a href="#"><u>VFSUnregisterDefaultDirectory</u></a> before re-registering it with <code>VFSRegisterDefaultDirectory</code> .  The specified directory registered for a given file type is intended to be the “root” default directory. If a given default directory has one or more subdirectories, applications should also search those subdirectories for files of the appropriate type.
<b>Compatibility</b>	Implemented only if the <a href="#"><u>VFS Manager Feature Set</u></a> is present.
<b>See Also</b>	<a href="#"><u>VFSGetDefaultDirectory</u></a>

## **VFSRemoveFSLib**

<b>Purpose</b>	Remove a file system library from the library table, so that the VFS Manager can no longer use it.
<b>Declared In</b>	<code>VfsMgr.h</code>
<b>Prototype</b>	<code>Err VFSRemoveFSLib (UInt16 fsLibRefNum)</code>
<b>Parameters</b>	-> <code>fsLibRefNum</code> Library reference number of the file system library to remove from the library table.
<b>Result</b>	Returns one of the following error codes:  <code>errNone</code> No error.  <code>expErrNotOpen</code> The file system library necessary for this call has not been installed or has not been opened.  <code>vfsErrNoFileSystem</code> VFS Manager can not find the file system specified in <code>fsLibRefNum</code> .

**Comments** This function is not normally called by applications. It unmounts any volumes that the specified file system may have mounted. It then closes the library and removes it from the library table with [SysLibRemove](#).

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

## **VFSUnregisterDefaultDirectory**

**Purpose** Sever the association between a particular file type and a default directory for a given type of card media.

**Declared In** VfsMgr.h

**Prototype** Err VFSUnregisterDefaultDirectory  
(const Char \*fileTypeStr, UInt32 mediaType)

**Parameters** -> fileTypeStr Pointer to the file type with which the default directory is associated. This is a null-terminated string that can either be a MIME media type/subtype pair, such as "image/jpeg", "text/plain", or "audio/basic"; or a file extension, such as ".jpeg".

-> mediaType Type of card media for which the default directory is associated. See [Defined Media Types](#) in the [Expansion Manager](#) chapter for the list of accepted values.

**Result** Returns one of the following error codes:

errNone No error.

sysErrParamErr The fileTypeStr parameter is NULL.

vfsErrFileNotFound

A default directory could not be found in the registry for the specified file and media type.

## Virtual File System Manager

### VFS Manager Functions

---

<b>Comments</b>	<b>NOTE:</b> Caution is advised when using this function, since you may remove another application's registration, causing data to mysteriously disappear from those applications.
<b>Compatibility</b>	Implemented only if the <a href="#">VFS Manager Feature Set</a> is present.
<b>See Also</b>	<a href="#">VFSGetDefaultDirectory</a> , <a href="#">VFSRegisterDefaultDirectory</a>

## VFSVolumeEnumerate

<b>Purpose</b>	Enumerate the mounted volumes.				
<b>Declared In</b>	<code>VfsMgr.h</code>				
<b>Prototype</b>	<code>Err VFSVolumeEnumerate (UInt16 *volRefNumP,                           UInt32 *volIteratorP)</code>				
<b>Parameters</b>	<p><code>&lt;- volRefNumP</code> Pointer to the reference number for the volume represented by the current enumeration, or <code>vfsInvalidVolRef</code> if there are no more volumes to be enumerated or an error occurred.</p> <p><code>&lt;-&gt; volIteratorP</code> Pointer to a variable that holds the index of the current enumeration. Set the variable to <code>vfsIteratorStart</code> prior to the first iteration. Each call to <code>VFSVolumeEnumerate</code> updates the variable to the index of the next volume. When the last volume is reached, the variable pointed to by <code>volIteratorP</code> is set to <code>vfsIteratorStop</code>.</p>				
<b>Result</b>	Returns one of the following error codes:  <table><tr><td><code>errNone</code></td><td>No error</td></tr><tr><td><code>expErrEnumerationEmpty</code></td><td>There are no volumes to enumerate.</td></tr></table>	<code>errNone</code>	No error	<code>expErrEnumerationEmpty</code>	There are no volumes to enumerate.
<code>errNone</code>	No error				
<code>expErrEnumerationEmpty</code>	There are no volumes to enumerate.				

**sysErrParamErr** The value pointed to by volIteratorP is not valid. This error is also returned when volIteratorP is vfsIteratorStop.

**Comments** This function returns a pointer to the volume reference number in the volRefNumP parameter. In order to traverse all volumes you must make repeated calls to [VFSVolumeEnumerate](#) inside a loop. Before the first call to VFSVolumeEnumerate, the variable pointed to by volIteratorP should be initialized to vfsIteratorStart. Each iteration then increments volIteratorP to the next entry after updating volRefNumP. When the last volume is reached, \*volIteratorP is set to vfsIteratorStop. If there are no volumes to enumerate, VFSVolumeEnumerate returns expErrEnumerationEmpty when first called.

**Example** Below is an example of how to use VFSVolumeEnumerate.

---

```
UInt16 volRefNum;
UInt32 volIterator = vfsIteratorStart;

while (volIterator != vfsIteratorStop) {
    err = VFSVolumeEnumerate(&volRefNum, &volIterator);
    if (err == errNone) {
        // Do something with the volRefNum
    } else {
        // handle error... possibly by
        // breaking out of the loop
    }
}
```

---

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

## **Virtual File System Manager**

### *VFS Manager Functions*

---

## **VFSVolumeFormat**

**Purpose** Format and mount the volume installed in a given slot.

**Declared In** VfsMgr.h

**Prototype** Err VFSVolumeFormat (UInt8 flags,  
                  UInt16 fsLibRefNum,  
                  VFSAnyMountParamPtr vfsMountParamP)

**Parameters**

-> flags	Flags that control how the volume should be formatted. Currently, the only flag not reserved is <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.
-> fsLibRefNum	Reference number of the file system library for which the volume should be formatted. This number is obtained through a call to <a href="#">SysLibFind</a> with the name of the library you want to use. If the <code>flags</code> field is not set to <code>vfsMountFlagsUseThisFileSystem</code> , this parameter is ignored.

<->vfsMountParamP

Parameters to be used when formatting the volume and when mounting the volume after it has been formatted. Supply a pointer to either a [VFSSlotMountParamType](#) or a [VFSPOSEMountParamType](#) structure. Note that you'll need to cast your structure pointer to a [VFSAnyMountParamPtr](#). Set the `mountClass` field to the appropriate value: if you are mounting to an Expansion Manager slot, set `mountClass` to `VFSMountClass_SlotDriver` and initialize `slotLibRefNum` and `slotRefNum` to the appropriate values. See the descriptions of [VFSAnyMountParamType](#), [VFSSlotMountParamType](#), and [VFSPOSEMountParamType](#) for information on the fields that make up these data structures.

**Result** Returns one of the following error codes:

`errNone` No error.

`expErrNotEnoughPower`

There is insufficient battery power to format and/or mount a volume.

`expErrNotOpen` The file system library necessary for this call has not been installed or has not been opened.

`vfsErrNoFileSystem`

The VFS Manager cannot find an appropriate file system to handle the request.

**Comments**

The slot driver currently only supports one volume per slot. If the volume is successfully formatted and mounted, the reference number of the mounted volume is returned in `vfsMountParamP->volRefNum`. If the format is unsuccessful or cancelled, `vfsMountParamP->volRefNum` is set to `vfsInvalidVolRef`. If `vfsMountFlagsUseThisFileSystem` is passed as a flag, `VFSVolumeFormat` attempts to format the volume using the file system library specified by `fsLibRefNum`. Typically the flag

## **Virtual File System Manager**

### *VFS Manager Functions*

---

parameter is not set. In this case VFSVolumeFormat tries to find a compatible library to format the volume, as follows:

1. Check to see if the default file system library feature is set. If it is, and if that file system is installed, it is used to format the volume. You can set the default file system using [FtrSet](#); supply sysFileCVFSMgr for the feature creator, and vfsFtrIDDefaultFS for the feature number.
2. Check to see if any of the installed file systems are natively supported for the slot on which the VFS Manager is trying to format. If one of them is, it is used to format the volume.
3. If none of the installed file systems can perform the format using the slot's native type, a dialog displays warning the user that their media may become incompatible with other devices if they continue with the format. The user may continue or cancel the format. If the user chooses to continue, VFSVolumeFormat formats the volume using the first file system library that was installed.

When calling VFSVolumeFormat, the volume can either be mounted or unmounted. The underlying file system library call requires the volume to be unmounted. VFSVolumeFormat checks to see if the volume is currently mounted and unmounts it, if necessary, using [VFSVolumeUnmount](#) before making the file system call. If the file system successfully formats the volume, VFSVolumeFormat mounts it and posts a [sysNotifyVolumeMountedEvent](#) notification.

**Example** The following code excerpt formats a volume on an Expansion Manager slot using a compatible file system.

---

```
VFSSlotMountParamType slotParam;
UInt32 slotIterator = expIteratorStart;

slotParam.vfsMountParamP.mountClass =
    VFSMountClass_SlotDriver;
err = ExpSlotEnumerate(&slotParam.slotRefNum,
    &slotIterator);
err = ExpSlotLibFind(slotParam.slotRefNum,
    &slotParam.slotLibRefNum);

err = VFSVolumeFormat(NULL, NULL,
    (VFSAnyMountParamPtr) & slotParam);
```

---

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSVolumeMount](#)

## **VFSVolumeGetLabel**

**Purpose** Determine the volume label for a particular volume.

**Declared In** VfsMgr.h

**Prototype** Err VFSVolumeGetLabel (UInt16 volRefNum,  
Char \*labelP, UInt16 bufLen)

**Parameters** -> volRefNum Volume reference number returned from  
[VFSVolumeEnumerate](#).

<- labelP Pointer to a character buffer into which the  
volume name is placed.

-> bufLen Length, in bytes, of the labelP buffer.

**Result** Returns one of the following error codes:

errNone No error.

expErrNotOpen The file system library necessary for this call  
has not been installed or has not been opened.

vfsErrNoFileSystem

The VFS Manager cannot find an appropriate  
file system to handle the request.

vfsErrVolumeBadRef

The specified volume has not been mounted.

vfsErrBufferOverflow

The value specified in bufLen is not big  
enough to receive the full volume label.

vfsErrNameShortened

There was an error reading the full volume  
name. A shortened version is being returned.

## Virtual File System Manager

### VFS Manager Functions

---

<b>Comments</b>	Volume reference numbers can change each time you mount a given volume. To keep track of a particular volume, save the volume's label rather than its reference number. Volume labels can be up to 255 characters long. They can contain any normal character, including spaces and lower case characters, in any character set as well as the following special characters: \$ % ' - _ @ ~ ` ! ( ) ^ # & + , ; = [ ].
<b>Compatibility</b>	Implemented only if the <a href="#">VFS Manager Feature Set</a> is present.
<b>See Also</b>	<a href="#">VFSVolumeSetLabel</a>

## VFSVolumeInfo

<b>Purpose</b>	Get information about the specified volume.							
<b>Declared In</b>	VfsMgr.h							
<b>Prototype</b>	Err VFSVolumeInfo(UInt16 volRefNum, VolumeInfoType *volInfoP)							
<b>Parameters</b>	<table><tr><td>-&gt; volRefNum</td><td>Volume reference number returned from <a href="#">VFSVolumeEnumerate</a>.</td></tr><tr><td>&lt;- volInfoP</td><td>Pointer to the structure that receives the volume information for the specified volume. See <a href="#">VolumeInfoType</a> for more information on the fields in this data structure.</td></tr></table>		-> volRefNum	Volume reference number returned from <a href="#">VFSVolumeEnumerate</a> .	<- volInfoP	Pointer to the structure that receives the volume information for the specified volume. See <a href="#">VolumeInfoType</a> for more information on the fields in this data structure.		
-> volRefNum	Volume reference number returned from <a href="#">VFSVolumeEnumerate</a> .							
<- volInfoP	Pointer to the structure that receives the volume information for the specified volume. See <a href="#">VolumeInfoType</a> for more information on the fields in this data structure.							
<b>Result</b>	<p>Returns one of the following error codes:</p> <table><tr><td>errNone</td><td>No error.</td></tr><tr><td>expErrNotOpen</td><td>The file system library necessary for this call has not been installed or has not been opened.</td></tr><tr><td>vfsErrNoFileSystem</td><td>The VFS Manager cannot find an appropriate file system to handle the request.</td></tr></table>		errNone	No error.	expErrNotOpen	The file system library necessary for this call has not been installed or has not been opened.	vfsErrNoFileSystem	The VFS Manager cannot find an appropriate file system to handle the request.
errNone	No error.							
expErrNotOpen	The file system library necessary for this call has not been installed or has not been opened.							
vfsErrNoFileSystem	The VFS Manager cannot find an appropriate file system to handle the request.							

`vfsErrVolumeBadRef`

The specified volume reference number is invalid.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSVolumeGetLabel](#), [VFSVolumeSize](#)

## VFSVolumeMount

**Purpose** Mount the card's volume on the specified slot.

**Declared In** VfsMgr.h

**Prototype** Err VFSVolumeMount (UInt8 flags,  
                  UInt16 fsLibRefNum,  
                  VFSAnyMountParamPtr vfsMountParamP)

**Parameters**

-> flags	Flags that control how the volume should be mounted. Currently, the only flag not reserved is <code>vfsMountFlagsUseThisFileSystem</code> . Pass this flag to cause the volume to be mounted using the file system specified by <code>fsLibRefNum</code> . Pass zero (0) to have the VFS Manager attempt to mount the volume using a file system appropriate for the slot.
-> fsLibRefNum	Reference number of the file system library for which the volume should be mounted. This number is obtained through a call to <a href="#">SysLibFind</a> with the name of the library you want to use. If the <code>flags</code> field is not set to <code>vfsMountFlagsUseThisFileSystem</code> , this parameter is ignored.

## Virtual File System Manager

### VFS Manager Functions

---

<->vfsMountParamP

Parameters to be used when mounting the volume after it has been formatted. Supply a pointer to either a [VFSSlotMountParamType](#) or a [VFSPOSEMountParamType](#) structure. Note that you'll need to cast your structure pointer to a VFSAnyMountParamPtr. Set the mountClass field to the appropriate value: if you are mounting to an Expansion Manager slot, set mountClass to VFSMountClass\_SlotDriver and initialize slotLibRefNum and slotRefNum to the appropriate values. See the descriptions of [VFSAnyMountParamType](#), [VFSSlotMountParamType](#), and [VFSPOSEMountParamType](#) for information on the fields that make up these data structures.

**Result** Returns one of the following error codes:

errNone	No error.
expErrNotEnoughPower	There is insufficient battery power to mount a volume.
expErrNotOpen	The file system library necessary for this call has not been installed or has not been opened.
sysErrParamErr	vfsMountParamP was initialized to NULL.
vfsErrNoFileSystem	The VFS Manager cannot find an appropriate file system to handle the request.
vfsErrVolumeStillMounted	The volume is already mounted with a different file system than was specified in fsLibRefNum.

**Comments**

The slot driver only supports one volume per slot. The reference number of the mounted volume is returned in vfsMountParamP->volRefNum. If vfsMountFlagsUseThisFileSystem is passed

as a flag, VFSVolumeMount attempts to mount the volume using the file system library specified by `fsLibRefNum`. Otherwise VFSVolumeMount tries to find a file system library which is able to mount the volume. If none of the installed file system libraries is able to mount the volume, VFSVolumeMount attempts to re-format the volume (using [VFSVolumeFormat](#)) and then mount it. If VFSVolumeMount manages to successfully mount the volume, it ends by posting a [sysNotifyVolumeMountedEvent](#) notification.

After VFSVolumeMount successfully mounts a volume, it broadcasts `sysNotifyVolumeMountedEvent`. The VFS Manager, upon being notified of this event, searches the newly-mounted volume for `/PALM/start.prc`. If `start.prc` is found in the `/PALM` directory, the VFS Manager copies it to main memory and launches it. If `start.prc` is not found, the VFS Manager switches to the Launcher instead. This behavior can be overridden; see [Card Insertion and Removal](#) in [Chapter 7, “Expansion,” Palm OS Programmer’s Companion](#), vol. I.

When VFSVolumeMount is called, if the volume is already mounted with a different file system than was specified in `fsLibRefNum`, a `vfsErrVolumeStillMounted` error is returned. If the volume is already mounted with the same file system that is specified in `fsLibRefNum`, or if `vfsMountFlagsUseThisFileSystem` is not set, VFSVolumeMount returns `errNone` and sets `volRefNumP` to the reference number of the currently mounted volume.

**Example** The following code excerpt mounts a volume on an Expansion Manager slot using a compatible file system.

---

```
VFSSlotMountParamType slotParam ;
UInt32 slotIterator = expIteratorStart;

slotParam.vfsMountParamP.mountClass =
    VFSMountClass_SlotDriver;
err = ExpSlotEnumerate(&slotParam.slotRefNum,
    &slotIterator);
err = ExpSlotLibFind(slotParam.slotRefNum,
    &slotParam.slotLibRefNum);

err = VFSVolumeMount(NULL, NULL,
    (VFSAnyMountParamPtr) & slotParam);
```

---

## **Virtual File System Manager**

### *VFS Manager Functions*

---

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSVolumeFormat](#), [VFSVolumeUnmount](#)

## **VFSVolumeSetLabel**

**Purpose** Changes the volume label for a mounted volume.

**Declared In** VfsMgr.h

**Prototype** Err VFSVolumeSetLabel (UInt16 volRefNum,  
const Char \*labelP)

**Parameters** -> volRefNum Volume reference number returned from  
[VFSVolumeEnumerate](#).

-> labelP Pointer to the label to be applied to the  
specified volume. This string must be null-  
terminated.

**Result** Returns one of the following error codes:

errNone No error

expErrNotOpen The file system library necessary for this call  
has not been installed or has not been opened.

vfsErrBadName The supplied label is invalid.

vfsErrNameShortened Indicates that the label name was too long. A  
shortened version of the label name was used  
instead.

vfsErrVolumeBadRef The specified volume has not been mounted.

**Comments** Volume labels can be up to 255 characters long. They can contain  
any normal character, including spaces and lower case characters, in  
any character set as well as the following special characters: \$ % ' - \_  
@ ~ ` ! ( ) ^ # & + , ; = [ ]. See [Naming Volumes](#) in [Chapter 7](#),

"[Expansion](#)," *Palm OS Programmer's Companion*, vol. I for guidelines on naming.

---

**NOTE:** Most clients should not need to call this function. This function may create or delete a file in the root directory, which would invalidate any current calls to [VFSDirEntryEnumerate](#).

---

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSVolumeGetLabel](#)

## VFSVolumeSize

**Purpose** Determine the total amount of space on a volume, as well as the amount that is currently being used.

**Declared In** VfsMgr.h

**Prototype** Err VFSVolumeSize (UInt16 volRefNum,  
                  UInt32 \*volumeUsedP, UInt32 \*volumeTotalP)

**Parameters**

-> volRefNum	Volume reference number returned from <a href="#">VFSVolumeEnumerate</a> .
<- volumeUsedP	Pointer to a variable that receives the amount of space, in bytes, in use on the volume.
<- volumeTotalP	Pointer to a variable that receives the total amount of space on the volume, in bytes.

**Result** Returns one of the following error codes:

errNone No error.

expErrNotOpen The file system library necessary for this call has not been installed or has not been opened.

## **Virtual File System Manager**

### *VFS Manager Functions*

---

`vfsErrNoFileSystem`

The VFS Manager cannot find an appropriate file system to handle the request.

`vfsErrVolumeBadRef`

The specified volume has not been mounted.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSVolumeInfo](#)

## **VFSVolumeUnmount**

**Purpose** Unmount the given volume.

**Declared In** `VfsMgr.h`

**Prototype** `Err VFSVolumeUnmount (UInt16 volRefNum)`

**Parameters** `-> volRefNum` Volume reference number returned from [VFSVolumeEnumerate](#).

**Result** Returns one of the following error codes:

`errNone` No error.

`expErrNotOpen` The file system library necessary for this call has not been installed or has not been opened.

`vfsErrNoFileSystem`

The VFS Manager cannot find an appropriate file system to handle the request.

`vfsErrVolumeBadRef`

The specified volume has not been mounted.

**Comments** This function closes any opened files and posts a [sysNotifyVolumeUnmountedEvent](#) notification once the file system is successfully unmounted.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSSVolumeMount](#)

## Application-Defined Functions

### **VFSExportProcPtr**

**Purpose** User-defined callback function supplied to [VFSExportDatabaseToFileCustom](#) that tracks the progress of the export.

**Declared In** VfsMgr.h

**Prototype**

```
typedef Err (*VFSExportProcPtr)
(UInt32 totalBytes, UInt32 offset,
void *userDataP)
```

**Parameters** **totalBytes** The total number of bytes being exported.

**offset** Undefined.

**userDataP** Pointer to any application-specific data passed to the callback function. This pointer may be NULL if your callback doesn't need any such data.

**Result** Your progress tracker should allow the user to abort the export. Return errNone if the export should continue, or any other value to abort the export process. If you return a value other than errNone, that value will be returned by [VFSExportDatabaseToFileCustom](#).

**Comments** See the [Progress Dialogs](#) section in the *Palm OS Programmer's Companion*, vol. I for more information on writing a progress tracker.

**Compatibility** Implemented only if the [VFS Manager Feature Set](#) is present.

**See Also** [VFSSImportProcPtr](#)

## VFSImportProcPtr

<b>Purpose</b>	User-defined callback function supplied to <a href="#">VFSImportDatabaseFromFileCustom</a> that tracks the progress of the import.						
<b>Declared In</b>	VfsMgr.h						
<b>Prototype</b>	<pre>typedef Err (*VFSImportProcPtr) (UInt32 totalBytes, UInt32 offset, void *userDataP)</pre>						
<b>Parameters</b>	<table><tr><td>totalBytes</td><td>The total number of bytes being imported.</td></tr><tr><td>offset</td><td>The number of bytes that have already been imported. This value, along with the total number of bytes being imported, allows you to inform the user how far along the import is.</td></tr><tr><td>userDataP</td><td>Pointer to NY application-specific data passed to the callback function. This pointer may be NULL if your callback doesn't need any such data.</td></tr></table>	totalBytes	The total number of bytes being imported.	offset	The number of bytes that have already been imported. This value, along with the total number of bytes being imported, allows you to inform the user how far along the import is.	userDataP	Pointer to NY application-specific data passed to the callback function. This pointer may be NULL if your callback doesn't need any such data.
totalBytes	The total number of bytes being imported.						
offset	The number of bytes that have already been imported. This value, along with the total number of bytes being imported, allows you to inform the user how far along the import is.						
userDataP	Pointer to NY application-specific data passed to the callback function. This pointer may be NULL if your callback doesn't need any such data.						
<b>Result</b>	Your progress tracker should allow the user to abort the import. Return errNone if the import should continue, or any other value to abort the import process. If you return a value other than errNone, that value will be returned by <a href="#">VFSImportDatabaseFromFileCustom</a> .						
<b>Comments</b>	See the <a href="#">Progress Dialogs</a> section in the <i>Palm OS Programmer's Companion</i> , vol. I for more information on writing a progress tracker.						
<b>Compatibility</b>	Implemented only if the <a href="#">VFS Manager Feature Set</a> is present.						
<b>See Also</b>	<a href="#">VFSExportProcPtr</a>						

# Windows

---

This chapter provides information about windows by discussing these topics:

- [Window Data Structures](#)
- [Window Constants](#)
- [Window Functions](#)

No resources are associated with window objects.

The header file `Window.h` declares the API that this chapter describes. For more information on windows, see the section “[Text](#)” in the *Palm OS Programmer’s Companion*, vol. I.

## Window Data Structures

### CustomPatternType

The `CustomPatternType` type holds an 8-by-8 bit pattern that is one bit deep. Each byte specifies a row of the pattern. When drawing, a pattern is tiled to fill a specified region. This pattern is used by [WinFillLine](#) and [WinFillRectangle](#).

The [PatternType](#) specifies the name of the current pattern.

```
typedef UInt8 CustomPatternType [8];
```

#### Compatibility

In pre-3.5 systems, the `CustomPatternType` is an array of 4 16-bit words. Note the size of this data type has not changed.

### DrawStateType

The `DrawStateType` structure defines the current drawing state, which is the Palm OS® implementation of a **pen**. This drawing state is saved with [WinPushDrawState](#) and restored with [WinPopDrawState](#).

## Windows

### Window Data Structures

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `DrawStateType` structure. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct DrawStateType {
    WinDrawOperation transferMode;
    PatternType pattern;
    UnderlineModeType underlineMode;
    FontID fontId;
    FontPtr font;
    CustomPatternType patternData;
    IndexedColorType foreColor;
    IndexedColorType backColor;
    IndexedColorType textColor;
    UInt8 reserved;
    RGBColorType foreColorRGB;
    RGBColorType backColorRGB;
    RGBColorType textColorRGB;
    UInt16 coordinateSystem;
    UInt16 reserved2;
    Fixed16 scale;
    Fixed16 ntvToActiveScale;
    Fixed16 stdToActiveScale;
    Fixed16 activeToStdScale;
} DrawStateType;
```

## Field Descriptions

transferMode	The current transfer mode for color drawing. See <a href="#">WinDrawOperation</a> . Use <a href="#">WinSetDrawMode</a> to set this value.
pattern	The name of the current pattern. See <a href="#">PatternType</a> . If set to customPattern, the patternData field contains the actual pattern. Use <a href="#">WinGetPatternType</a> and <a href="#">WinSetPatternType</a> to retrieve and set this value.
underlineMode	The current underline mode. See <a href="#">UnderlineModeType</a> . Use <a href="#">WinSetUnderlineMode</a> to set this value.
fontId	The ID of the current font. Use <a href="#">FntSetFont</a> to set this value.
font	A pointer to the current font. Use <a href="#">FntSetFont</a> to set this value.
patternData	The current pattern being used by the WinFill functions if pattern is customPattern. See <a href="#">CustomPatternType</a> . Use <a href="#">WinGetPattern</a> and <a href="#">WinSetPattern</a> to retrieve and set this value.
foreColor	Index of the current color used for the foreground. Use <a href="#">WinSetForeColor</a> to set this value. This field is only valid for indexed color bitmaps.
backColor	Index of the current color used for the background. Use <a href="#">WinSetBackColor</a> to set this value. This field is only valid for indexed color bitmaps.
textColor	Index of the current color used for text. Use <a href="#">WinSetTextColor</a> to set this value. This field is only valid for indexed color bitmaps.

## Windows

### Window Data Structures

---

reserved	Reserved for future use.
foreColorRGB	RGB value of the current color used for the foreground. Use <a href="#">WinSetForeColorRGB</a> to set this value. This field is only valid for Palm OS 4.0 or later, and only valid for direct color bitmaps.
backColorRGB	RGB value of the current color used for the background. Use <a href="#">WinSetBackColorRGB</a> to set this value. This field is only valid for Palm OS 4.0 or later, and only valid for direct color bitmaps.
textColorRGB	RGB value of the current color used for text. Use <a href="#">WinSetTextColorRGB</a> to set this value. This field is only valid for Palm OS 4.0 or later, and only valid for direct color bitmaps.
coordinateSystem	Active coordinate system.
reserved2	Reserved for future use.
scale	A fixed point value used to convert from the draw window's active coordinate system to native coordinates. This field is defined only if the <a href="#">High-Density Display Feature Set</a> is present.
ntvToActiveScale	A fixed point value used to convert from the native coordinate system to the draw window's active coordinate system. This field is defined only if the <a href="#">High-Density Display Feature Set</a> is present.

`stdToActiveScale` A fixed point value used to convert from the standard coordinate system to the draw window's active coordinate system. This field is used internally to convert font metrics, which are stored as standard coordinates. This field is defined only if the [High-Density Display Feature Set](#) is present.

`activeToStdScale` The inverse of `stdToActive`; the active-to-standard scaling factor. This field is defined only if the [High-Density Display Feature Set](#) is present.

<b>Compatibility</b>	This type is implemented only if <a href="#">3.5 New Feature Set</a> is present. The <code>scale</code> , <code>ntvToActiveScale</code> , <code>stdToActiveScale</code> , and <code>activeToStdScale</code> fields are defined only if the <a href="#">High-Density Display Feature Set</a> is present.
----------------------	---

## FrameBitsType

The `FrameBitsType` structure specifies attributes of a window's frame.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `FrameBitsType` bit field. Never access its bit field members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef union FrameBitsType {
    struct {
        UInt16 cornerDiam : 8;
        UInt16 reserved_3 : 3;
        UInt16 threeD : 1;
        UInt16 shadowWidth : 2;
        UInt16 width : 2;
    } bits;
    UInt16 word;
} FrameBitsType;
```

## Windows

### Window Data Structures

---

#### Field Descriptions

cornerDiam	Corner radius of frame; maximum is 38.
reserved_3	Reserved.
threeD	Set this bit to draw a 3D button. This feature is not currently supported.
shadowWidth	Width of shadow.
width	Frame width.
word	Reserved.

## FrameType

The FrameType type specifies a window frame style.

```
typedef UInt16 FrameType;
```

The FrameType can be set to one of the defined frame types shown in the table below, or a custom frame type as defined by a [FrameBitsType](#) structure.

Constant	Value	Description
noFrame	0	No frame
simpleFrame	1	Plain rectangular frame
rectangleFrame	1	Plain rectangular frame
simple3DFrame	0x0012	3D frame with width of 2. This frame type is not supported.
roundFrame	0x0401	Round frame with width of 1.
boldRoundFrame	0x0702	Round frame with width of 2.
popupFrame	0x0205	Popup frame style with slight corner roundness, width of 1 and shadow of 1.
dialogFrame	0x0302	Dialog frame style with slight corner roundness and width of 2.
menuFrame	popupFrame	Same as popupFrame .

## IndexedColorType

The `IndexedColorType` type is used to specify a color by its index value; that is, by its location in a color table. Color tables are defined by the [ColorTableType](#) structure, which is declared in `Bitmap.h`. The `IndexedColorType` can hold a 1, 2, 4, or 8-bit index.

```
typedef UInt8 IndexedColorType;
```

**Compatibility** This type is implemented only if [3.5 New Feature Set](#) is present.

## PatternType

The `PatternType` enumerated type specifies a pattern for drawing. This type is returned by [WinGetPatternType](#) and is used as a parameter to the [WinSetPatternType](#) function.

```
typedef enum {
    blackPattern,
    whitePattern,
    grayPattern,
    customPattern,
    lightGrayPattern,
    darkGrayPattern
} PatternType;
```

### Value Descriptions

<code>blackPattern</code>	Pattern with all bits on.
<code>whitePattern</code>	Pattern with all bits off.
<code>grayPattern</code>	Pattern with alternating on and off bits.
<code>customPattern</code>	Custom pattern specified by <a href="#">CustomPatternType</a> .

## Windows

### Window Data Structures

---

`lightGrayPattern` Pattern with one out of four bits in each row turned on. This value is defined only if the [High-Density Display Feature Set](#) is present.

`darkGrayPattern` Pattern with one out of four bits in each row turned off. This value is defined only if the [High-Density Display Feature Set](#) is present.

These patterns all operate with current foreground and background color instead of black and white. In effect, `blackPattern` is only black if the current foreground color is black. `whitePattern` uses the current background color. `grayPattern` and `customPattern` uses a combination of background and foreground colors.

Patterns are expanded to the destination bit depth by the blitter when drawing patterned lines and filled rectangles.

The three standard gray patterns—`grayPattern`, `lightGrayPattern`, and `darkGrayPattern`—are always drawn by the blitter using the screen density to improve the appearance of gray fills. Custom patterns, however, are stretched as appropriate by the blitter based on the destination density.

#### Compatibility

The `lightGrayPattern` and `darkGrayPattern` values are defined only if the [High-Density Display Feature Set](#) is present.

## UnderlineModeType

The `UnderlineModeType` enumerated type specifies possible values for the underline mode stored in [DrawStateType](#).

```
typedef enum { noUnderline, grayUnderline,  
solidUnderline, colorUnderline }  
UnderlineModeType;
```

### Value Descriptions

noUnderline	No underline.
grayUnderline	Underline is drawn using a dotted line in the current foreground color.
solidUnderline	Underline is drawn using a solid line in the foreground color.
colorUnderline	Underline is drawn using a solid line in the foreground color.

### Compatibility

The solidUnderline and colorUnderline options are only available in Palm OS 3.1 and higher.

## WindowFlagsType

The WindowFlagsType specifies different window attributes.

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the WindowFlagsType bit field. Access it only through the functions described below. Never access its bit field members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct WindowFlagsType {  
    UInt16 format:1;  
    UInt16 offscreen:1;  
    UInt16 modal:1;  
    UInt16 focusable:1;  
    UInt16 enabled:1;  
    UInt16 visible:1;  
    UInt16 dialog:1;  
    UInt16 freeBitmap:1;
```

## Windows

### Window Data Structures

---

```
    UInt16 reserved :8;  
} WindowFlagsType;
```

#### Field Descriptions

format	If set, use the genericFormat. If 0, use screenFormat. Screen format is the native format of the video system; windows in this format can be copied to the display faster. The generic format is device-independent. A window cannot be enabled (that is, accept pen input) unless it uses screen format.
offscreen	If set, the window is offscreen. If 0, the window is onscreen.
modal	If set, the window is modal. If 0, the window is not modal. You set this value when you create the window. This value is returned by <a href="#">WinModal</a> .
focusable	If set, the window can accept the focus. If 0, the window does not accept the focus. You set this value when you create the window.
enabled	If set, the window is enabled. If 0, the window is disabled.
visible	If set, the window is visible if it is onscreen. If 0, the window is not visible.
dialog	If set, the window is a form. If 0, the window is not a form. The <a href="#">FrmInitForm</a> function sets this value.
freeBitmap	If set, free the bitmap when the window is freed. If 0, retain the bitmap after the window is freed.
reserved	Reserved for future use. Must be 0.

#### Compatibility

In OS versions previous to 3.5, the freeBitmap flag was not present. Instead, a compressed flag was present, where 0 specified uncompressed and 1 specified compressed. This compressed flag is now part of the [BitmapType](#).

## **WindowFormatType**

Use this enumeration to specify the window format when creating an offscreen window with the [WinCreateOffscreenWindow](#) function.

```
typedef enum {
    screenFormat = 0,
    genericFormat,
    nativeFormat
} WindowFormatType;
```

### **Field Values**

screenFormat	The window's bitmap is allocated using the hardware screen's depth, but for backward compatibility the bitmap associated with the offscreen window is always low density, and the window always uses a coordinate system that directly maps offscreen pixels to coordinates.
genericFormat	Like screenFormat, except that genericFormat offscreen windows do not accept pen input.
nativeFormat	Reflects the actual hardware screen format in all ways, including screen depth, density, and pixel format. Applications must always use the graphic API when drawing to a nativeFormat offscreen window: directly accessing offscreen pixels will produce undefined results. When using this format, the width and height arguments must be specified using the active coordinate system. Like genericFormat, nativeFormat offscreen windows do not accept pen input.

<b>Compatibility</b>	WindowFormatType is defined only if the <a href="#">High-Density Display Feature Set</a> is present.
----------------------	--

## **WindowType**

The WindowType structure represents a window.

## Windows

### Window Data Structures

---

**WARNING!** PalmSource, Inc. does not support or provide backward compatibility for the `WindowType` structure. Access it only through the functions described below. Never access its structure members directly, or your code may break in future versions. Use the information below for debugging purposes only.

---

```
typedef struct WindowType {
    Coord           displayWidthV20;
    Coord           displayHeightV20;
    void            *displayAddrV20;
    WindowFlagsType windowFlags;
    RectangleType   windowBounds;
    AbsRectType     clippingBounds;
    BitmapPtr       bitmapP;
    FrameBitsType   frameType;
    DrawStateType   *drawStateP;
    struct WindowType *nextWindow;
} WindowType;
```

#### Field Descriptions

`displayWidthV20` Width of the window in pre OS 3.5 devices. In OS 3.5, use [WinGetBounds](#) to return the window width.

`displayHeightV20` Height of the window in pre OS 3.5 devices. In OS 3.5, use [WinGetBounds](#) to return the window height.

`displayAddrV20` Pointer to the window display memory buffer in pre OS 3.5 devices. In OS 3.5 or later, call [WinGetBitmap](#) and then [BmpGetBits](#) to obtain the window's memory buffer.

---

**WARNING!** Writing directly to screen memory will not be supported in all devices.

---

`windowFlags` Window attributes (see [WindowFlagsType](#)).

windowBounds	Display-relative bounds of the window. Use <a href="#">WinGetBounds</a> and <a href="#">WinSetBounds</a> to retrieve and set this value.
clippingBounds	Bounds for clipping any drawing within the window. Use <a href="#">WinGetClip</a> and <a href="#">WinSetClip</a> to retrieve and set this value.
bitmapP	Pointer to the window bitmap, which holds the window's contents. Use <a href="#">WinGetBitmap</a> to retrieve this value.
frameType	Frame attributes; see <a href="#">FrameBitsType</a> .
drawStateP	Pointer to a state of the current transfer mode, pattern mode, font, underline mode, and colors. See <a href="#">DrawStateType</a> . Only one drawing state exists in the system. Each window points to the same structure.
nextWindow	Pointer to the next window in a linked list of windows. This linked list of windows is called the active window list. Do not change this field.

### Compatibility

In OS versions previous to 3.5, this structure is slightly different. Specifically, the bitmapP field is instead a viewOrigin field of type PointType and specified the window origin point on the display. The drawStateP was named gstate and was of type GraphicStatePtr. The complete definition is shown below:

```
typedef struct WinTypeStruct {
    Word           displayWidth;
    Word           displayHeight;
    VoidPtr        displayAddr;
    WindowFlagsType windowFlags;
    RectangleType windowBounds;
    AbsRectType   clippingBounds;
    PointType      viewOrigin;
    FrameBitsType  frameType;
    GraphicStatePtr gstate;
    struct WinTypeStruct *nextWindow;
```

```
} WindowType;
```

## WinDrawOperation

The WinDrawOperation enumerated type specifies the transfer mode for color drawing. This type is used as a parameter to the [WinCopyRectangle](#) and [WinSetDrawMode](#) functions.

```
typedef enum {winPaint, winErase, winMask,  
winInvert, winOverlay, winPaintInverse,  
winSwap} WinDrawOperation;
```

### Value Descriptions

winPaint	Write color-matched source pixels to the destination. If a bitmap's hasTransparency flag is set, winPaint behaves like winOverlay instead.
winErase	Write backColor if the source pixel is transparent.
winMask	Write backColor if the source pixel is not transparent.
winInvert	Bitwise XOR the color-matched source pixel onto the destination. This mode does not honor the transparent color in any way.
winOverlay	Write color-matched source pixel to the destination if the source pixel is not transparent. Transparent pixels are skipped. For a 1-bit display, the "off" bits are considered to be the transparent color. Note that this definition of winOverlay is new for Palm OS 5: in Palm OS 4.x, the destination is set (only) where the source pixels are "on."

winPaintInverse	Invert the source pixel color and then proceed as with <code>winPaint</code> .
winSwap	Swap the <code>backColor</code> and <code>foreColor</code> destination colors if the source is a pattern (the type of pattern is disregarded). If the source is a bitmap, then the bitmap is transferred using <code>winPaint</code> mode instead.

Note that 2-bit, 4-bit, and 8-bit source bitmaps that don't have a color table inherit the system default color table for their given depth. 1-bit sources (bitmap, text, and patterns) that don't have a color table are given a color table where entry 0 is the `backColor` and entry 1 is the `foreColor` (`textColor` for text).

`winSwap` is not a color invert operation, although a pair of `winSwap` operations will restore the original graphics data. This mode is used by the OS to select and deselect areas of the screen. It changes destination pixels matching the foreground color to the background color, and changes destination pixels matching the background color to the foreground color. It is a mode available for rectangles, lines, and pixels, but not text or bitmaps. This mode ignores the current pattern. **The Transparent Color**

As of Palm OS 4.0, bitmaps have a `hasTransparency` flag and may designate a transparent color. These concepts are augmented somewhat in Palm OS 5 to make the transfer modes more consistent:

- Bitmaps that don't specify any transparent color (text, patterns, and version 0 bitmaps) are assumed to have a transparent color of index 0 and the `hasTransparency` bit is assumed to be false.
- When the `hasTransparency` flag is set and the transfer mode is `winPaint`, only the non-transparent pixels are copied to the destination. With text and patterns, the Palm OS assumes that the "off" bits are the ones designated as transparent and acts as if the `hasTransparency` flag is always false. This assumption retains backwards compatibility and unifies the use of transparency across all source data.

**Compatibility** This type is implemented only if [3.5 New Feature Set](#) is present. In earlier releases, this type is named ScrOperation and its values begin with the prefix scr rather than win. WinDrawOperation is fully compatible with ScrOperation. Transparency is only available if [4.0 New Feature Set](#) is present, and as mentioned in “[winSwap is not a color invert operation, although a pair of winSwap operations will restore the original graphics data. This mode is used by the OS to select and deselect areas of the screen. It changes destination pixels matching the foreground color to the background color, and changes destination pixels matching the background color to the foreground color. It is a mode available for rectangles, lines, and pixels, but not text or bitmaps. This mode ignores the current pattern. The Transparent Color](#)” the behavior of the window drawing modes—winOverlay in particular—changes slightly if [5.0 New Feature Set](#) is present.

## WinHandle

The WinHandle type is a pointer to a [WindowType](#) structure. Note that this may change.

```
typedef WindowType *WinHandle;
```

## WinLineType

The WinLineType structure defines a line.

```
typedef struct WinLineType {
    Coord x1;
    Coord y1;
    Coord x2;
    Coord y2;
} WinLineType;
```

### Field Descriptions

- |    |  |
|----|--|
| x1 | X coordinate of the first endpoint of the line.  |
| y1 | Y coordinate of the first endpoint of the line.  |
| x2 | X coordinate of the second endpoint of the line. |
| y2 | Y coordinate of the second endpoint of the line. |

**Compatibility** This type is implemented only if [3.5 New Feature Set](#) is present.

## WinPtr

The WinPtr type is a pointer to a [WindowType](#) structure.

```
typedef WindowType *WinPtr;
```

# Window Constants

If the [5.0 New Feature Set](#) or the [High-Density Display Feature Set](#) are present, the following constants are defined:

- [Window Coordinate System Constants](#)



## Window Coordinate System Constants

These constants specify the coordinate system to be used when drawing within a given window:

Constant	Value	Description
kCoordinatesNative	0	Use the bitmap's native coordinate system; this enables a 1-to-1 correspondence between coordinates and pixels.
kCoordinatesStandard	72	The coordinate system used by most handhelds running Palm OS 4.0 and earlier. On a single-density handheld, there is one screen pixel per standard coordinate. On a high-density screen, there is more than one screen pixel per standard coordinate.
kCoordinatesOneAndAHalf	108	One and a half times the standard coordinate system.
kCoordinatesDouble	144	Twice the standard coordinate system.
kCoordinatesTriple	216	Three times the standard coordinate system.
kCoordinatesQuadruple	288	Four times the standard coordinate system.

## Windows

### Window Functions

---

**IMPORTANT:** Not all coordinate systems listed in the above table are supported in this version of the [High-Density Display Feature Set](#). For Palm OS 5, only kCoordinatesNative, kCoordinatesStandard, and kCoordinatesDouble are supported.

---

Pass one of these constants as an argument to [WinSetCoordinateSystem](#). These values are returned by [WinGetCoordinateSystem](#).

**Compatibility** Defined only if the [High-Density Display Feature Set](#) is present.

## Window Functions

### WinClipRectangle

**Purpose** Shrink the rectangle to make it fit within the clipping region of the current draw window.

**Declared In** Window.h

**Prototype** void WinClipRectangle (RectangleType \*rP)

**Parameters** <-> rP Pointer to a structure holding the rectangle to clip. The rectangle returned is the intersection of the rectangle passed and the clipping bounds of the draw window.

**Result** Returns nothing.

**Comments** This function does not change the clipping rectangle of the window. To modify the window's clipping rectangle, use the [WinSetClip](#) and [WinResetClip](#) functions.

The draw window is the window to which all drawing functions send their output. It is returned by [WinGetDrawWindow](#).

**See Also** [WinCopyRectangle](#), [WinDrawRectangle](#),  
[WinEraseRectangle](#), [WinGetClip](#)

## WinCopyRectangle

**Purpose** Copy a rectangular region from one place to another (either between windows or within a single window).

**Declared In** Window.h

**Prototype** void WinCopyRectangle (WinHandle srcWin,  
WinHandle dstWin, const RectangleType \*srcRect,  
Coord destX, Coord destY, WinDrawOperation mode)

**Parameters**

-> srcWin	Window from which the rectangle is copied. If NULL, use the draw window.
-> dstWin	Window to which the rectangle is copied. If NULL, use the draw window.
-> srcRect	Bounds of the region to copy.
-> destX	Top bound of the rectangle in destination window.
-> destY	Left bound of the rectangle in destination window.
-> mode	The method of transfer from the source to the destination window (see <a href="#">WinDrawOperation</a> ).

**Result** Returns nothing.

**Comments** Copies the bits of the window inside the rectangle region.

If the destination bitmap is compressed, the mode parameter must be `winPaint`, and the destination coordinates must be (0,0). If the width of the destination rectangle is less than 16 pixels or if the

## Windows

### Window Functions

---

destination coordinates are not (0,0), then this function turns off compression for the destination bitmap. Normally, you do not copy to a compressed bitmap. Instead, you copy to an uncompressed bitmap and compress it afterwards.

**Compatibility** In OS versions before 3.5, the mode parameter was defined as type `ScrOperation`. It is defined as type `WinDrawOperation` only if [3.5 New Feature Set](#) is present. `ScrOperation` and `WinDrawOperation` are fully compatible with each other.

In OS versions before 3.5, it was common practice to render a bitmap in an offscreen window and then use `WinCopyRectangle` to draw it on the screen. In version 3.5 and higher, the preferred method of doing this is to use [WinDrawBitmap](#) or [WinPaintBitmap](#).

**See Also** [WinDrawBitmap](#)

## WinCreateBitmapWindow

**Purpose** Create a new offscreen window.

**Declared In** Window.h

**Prototype** WinHandle WinCreateBitmapWindow  
(BitmapType \*bitmapP, UInt16 \*error)

**Parameters** -> bitmapP Pointer to a bitmap to associate with the window. (See [BitmapType](#).)  
<- error Pointer to any error this function encounters.

**Result** Returns the handle of the new window upon success, or NULL if an error occurs. The error parameter contains one of the following:

errNone No error.

sysErrParamErr The `bitmapP` parameter is invalid. The bitmap must be uncompressed and it must have a valid pixel size (1, 2, 4, or 8). It must not be the screen bitmap.

sysErrNoFreeResource

There is not enough memory to allocate a new window structure.

**Comments**

Use `WinCreateBitmapWindow` if you want to draw into a previously created bitmap, such as a bitmap created using [BmpCreate](#).

This function generates a window wrapper for the specified bitmap. The newly created window is offscreen, uses the generic format (for device independence), and is added to the active window list. Use [WinSetDrawWindow](#) to make it the draw window, and then use the window drawing functions to modify the bitmap.

When you use this function to create a window and then delete the window with [WinDeleteWindow](#), the bitmap is **not** freed when the window is freed.

[WinCreateOffscreenWindow](#) uses this function to create its offscreen window. If you call `WinCreateOffscreenWindow` instead of using this function, the bitmap is freed when `WinDeleteWindow` is called.

The bitmap data will not be blitted properly if the depth of the screen is changed using [WinScreenMode](#) and the new window uses a bitmap that does not define the bitmap's color table. See [WinScreenMode](#) for information on how to work around this limitation.

**Compatibility**

Implemented only if [3.5 New Feature Set](#) is present.

**See Also**

[WinCreateWindow](#), [WinCreateOffscreenWindow](#)

## Windows

### Window Functions

---

## WinCreateOffscreenWindow

**Purpose** Create a new offscreen window and add it to the window list.

**Declared In** Window.h

**Prototype** WinHandle WinCreateOffscreenWindow (Coord width,  
Coord height, WindowFormatType format,  
UInt16 \*error)

**Parameters**

-> width	Width of the window in pixels. The coordinate system you use for this parameter depends upon the value of <code>format</code> .
-> height	Height of the window in pixels. The coordinate system you use for this parameter depends upon the value of <code>format</code> .
-> format	One of the window formats defined by <a href="#">WindowFormatType</a> .
<- error	Pointer to any error this function encounters.

**Result** Returns the handle of the new window upon success, or NULL if an error occurs. The `error` parameter contains one of the following:

errNone	No error.
sysErrParamErr	The <code>width</code> or <code>height</code> parameter is NULL or the current color table is invalid.
sysErrNoFreeResource	There is not enough memory to complete the function.

The debug ROM gives a warning if you try to draw to a bad window address.

**Comments** Windows created with this routine draw to a memory buffer instead of the display. Use this function for temporary drawing operations such as double-buffering or save-behind operations.

The memory buffer has two formats: screen format and generic format. Screen format is the native format of the video system;

windows in this format can be copied to the display faster. The generic format is device-independent. A window cannot be enabled (that is, accept pen input) unless it uses screen format.

This function differs from [WinCreateBitmapWindow](#) in the following ways:

- WinCreateOffscreenWindow creates a new bitmap in the same depth as the current screen.  
WinCreateBitmapWindow uses the bitmap you pass in, which may or may not be in the same depth as the current screen.
- WinCreateOffscreenWindow uses the screen format you specify. WinCreateBitmapWindow always uses genericFormat for the format argument.
- When you delete the window created with WinCreateOffscreenWindow, its bitmap is freed along with the window. The bitmap used in the WinCreateBitmapWindow is not freed when the window is freed.

Note that if you aren't directly accessing the bits of an offscreen window's bitmap but are just using the APIs, you can always pass nativeFormat for the screen format even on pre-Palm OS 5 handhelds and things will work as expected. If you need direct access to the bits of the offscreen window's bitmap, however, call [BmpCreate](#) and then call WinCreateBitmapWindow. Because you created the bitmap, you know its format and thus can safely manipulate its bits. Calling WinCreateOffscreenWindow with a format argument of nativeFormat can result in a bitmap with an unexpected format: the endianness, number of bits per pixel, and so on would match the screen and therefore be fastest to draw, but your application wouldn't be able to manipulate the pixels directly.

The bitmap data will not be blitted properly if the depth of the screen is changed using [WinScreenMode](#) and the new window uses a bitmap that does not define the bitmap's color table. See [WinScreenMode](#) for information on how to work around this limitation.

**See Also** [WinCreateWindow](#), [WinScreenMode](#),

## Windows

### Window Functions

---

## WinCreateWindow

**Purpose** Create a new window and add it to the window list.

**Declared In** Window.h

**Prototype** WinHandle WinCreateWindow  
(const RectangleType \*bounds, FrameType frame,  
Boolean modal, Boolean focusable, UInt16 \*error)

**Parameters**

-> bounds	Display-relative bounds of the window.
-> frame	Type of frame around the window (see <a href="#">FrameType</a> ).
-> modal	true if the window is modal.
-> focusable	true if the window can be the active window.
<- error	Pointer to any error encountered by this function.

**Result** Returns the handle of the new window upon success, or NULL if an error occurs. The error parameter contains one of the following:

errNone	No error.
sysErrNoFreeResource	There is not enough memory to complete the operation.

**Comments** Windows created by this routine draw to the display. See [WinCreateOffscreenWindow](#) for information on drawing off screen.

You typically don't call this function directly. Instead, you use [FrmInitForm](#) to create form windows from a resource. Forms are much more flexible and have better system support. All forms are windows, but not all windows are forms.

The window is created with the bounds and frame type that you specify and uses the bitmap and drawing state of the current draw window. Its clipping region is reset according to the bounds you specify.

All window flags are set to 0 except for the modal and focusable flags, which you pass as a parameter to this function. Specifically, newly created windows are disabled and invisible. You must specifically enable the window before the window can accept input. You can do so with [WinSetActiveWindow](#).

**See Also** [WinDeleteWindow](#)

## WinDeleteWindow

**Purpose** Remove a window from the window list and free the memory used by the window.

**Declared In** Window.h

**Prototype** void WinDeleteWindow (WinHandle winHandle,  
Boolean eraseIt)

**Parameters**

- > winHandle Handle of window to delete.
- > eraseIt If true, the window is erased before it is deleted. If false, the window is not erased.

**Result** Returns nothing.

**Comments** This function frees all memory associated with the window. Windows created using [WinCreateOffscreenWindow](#) have their bitmaps freed; windows created using [WinCreateWindow](#) or [WinCreateBitmapWindow](#) do not.

The eraseIt parameter affects onscreen windows only; offscreen windows are never erased. As a performance optimization, you might set eraseIt to false for an onscreen window if you know that you are going to immediately redraw the area anyway. For example, when the form manager closes a form dialog, it restores the area with the save-behind bits it had stored for that form. For this reason, when the form manager deletes the dialog window, it passes false for eraseIt because the entire area will be redrawn.

## Windows

### Window Functions

---

## WinDisplayToWindowPt

**Purpose** Convert a display-relative coordinate to a window-relative coordinate. The coordinate returned is relative to the display window.

**Declared In** Window.h

**Prototype** void WinDisplayToWindowPt (Coord \*extentX,  
Coord \*extentY)

**Parameters** <-> extentX Pointer to x coordinate to convert.  
<-> extentY Pointer to y coordinate to convert.

**Result** Returns nothing.

**See Also** [WinWindowToDisplayPt](#)

## WinDrawBitmap

**Purpose** Draw a bitmap at the given coordinates in winPaint mode (see [WinDrawOperation](#) for mode details).

**Declared In** Window.h

**Prototype** void WinDrawBitmap (BitmapPtr bitmapP, Coord x,  
Coord y)

**Parameters** -> bitmapP Pointer to a bitmap.  
-> x The x coordinate of the top-left corner.  
-> y The y coordinate of the top-left corner.

**Result** Returns nothing.

**Comments** If the bitmap has multiple depths (is a bitmap family), the closest match less than or equal to the current draw window depth is used.

If such a bitmap does not exist, the bitmap with the closest match greater than the draw window depth is used.

If the bitmap has its own color table, color conversion to the draw window color table will be applied (on OS 3.5 or later). This color conversion is slow and not recommended. Instead of including a color table in the bitmap, consider using [WinPalette](#) to change the system color table, draw the bitmap, and then change the system color table back when the bitmap is no longer visible.

This function differs from [WinPaintBitmap](#) in that this function always uses winPaint mode (copy mode) as the transfer mode. WinPaintBitmap uses the current drawing state transfer mode.

**See Also** [WinEraseRectangle](#)

## WinDrawChar

**Purpose** Draw the specified character in the draw window.

**Declared In** Window.h

**Prototype** void WinDrawChar (WChar theChar, Coord x,  
Coord y)

**Parameters**

-> theChar	The character to draw. This may be either a single-byte character or a multi-byte character.
-> x	x coordinate of the location where the character is to be drawn (left bound).
-> y	y coordinate of the location where the character is to be drawn (top bound).

**Result** Returns nothing.

**Comments** Before calling this function, call [WinSetUnderlineMode](#) and [FntSetFont](#) to set the desired underline and font to draw the characters.

This function differs from [WinPaintChar](#) in that this function always uses winPaint mode (see [WinDrawOperation](#)). This

## Windows

### Window Functions

---

means the on bits are drawn in the text color, the off bits are in the background color, and underlines are in the foreground color. `WinPaintChar` uses the current drawing state transfer mode instead of `winPaint`.

**Compatibility** Implemented only if [3.1 New Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the `PalmOSGlue` library and call `WinGlueDrawChar`. For more information, see [Chapter 75, “PalmOSGlue Library.”](#)

**See Also** [WinDrawChars](#), [WinDrawInvertedChars](#),  
[WinDrawTruncChars](#), [WinEraseChars](#), [WinInvertChars](#),  
[WinPaintChars](#)

## WinDrawChars

**Purpose** Draw the specified characters in the draw window.

**Declared In** `Window.h`

**Prototype** `void WinDrawChars (const Char *chars, Int16 len,  
Coord x, Coord y)`

**Parameters**

-> chars	Pointer to the characters to draw.
-> len	Length in bytes of the characters to draw.
-> x	x coordinate of the first character to draw (left bound).
-> y	y coordinate of the first character to draw (top bound).

**Result** Returns nothing.

**Comments** This function is useful for printing non-editable status or warning messages on the screen.  
Before calling this function, call [WinSetUnderlineMode](#) and [FntSetFont](#) to set the desired underline and font to draw the characters.

This function differs from [WinPaintChars](#) in that this function always uses winPaint mode (see [WinDrawOperation](#)). This means the on bits are drawn in the text color, the off bits are in the background color, and underlines are in the foreground color. WinPaintChar uses the current drawing state transfer mode instead of winPaint.

**See Also** [WinDrawChar](#), [WinDrawInvertedChars](#),  
[WinDrawTruncChars](#), [WinEraseChars](#), [WinInvertChars](#),  
[WinPaintChar](#)

## WinDrawGrayLine

**Purpose** Draw a dashed line in the draw window.

**Declared In** Window.h

**Prototype** void WinDrawGrayLine (Coord x1, Coord y1,  
Coord x2, Coord y2)

**Parameters**

-> x1	x coordinate of line start point.
-> y1	y coordinate of line start point.
-> x2	x coordinate of line endpoint.
-> y2	y coordinate of line endpoint.

**Result** Returns nothing.

**Comments** This routine does not draw in the gray color; it draws with alternating foreground and background pixels. That is, it uses the grayPattern pattern type.

**See Also** [WinDrawLine](#), [WinEraseLine](#), [WinFillLine](#), [WinInvertLine](#),  
[WinPaintLine](#), [WinPaintLines](#)

## Windows

### Window Functions

---

## WinDrawGrayRectangleFrame

**Purpose** Draw a gray rectangular frame in the draw window.

**Declared In** Window.h

**Prototype** void WinDrawGrayRectangleFrame (FrameType frame,  
const RectangleType \*rP)

**Parameters** -> frame Type of frame to draw (see [FrameType](#)).  
-> rP Pointer to the rectangle to frame.

**Result** Returns nothing.

**Comments** This routine does not draw in the gray color; it draws with alternating foreground and background pixels. The standard gray pattern is not used by this routine; rather, the frame is drawn so that the top-left pixel of the frame is always on.

**See Also** [WinDrawRectangleFrame](#), [WinEraseRectangleFrame](#),  
[WinGetFramesRectangle](#), [WinInvertRectangleFrame](#),  
[WinPaintRectangleFrame](#)

## WinDrawInvertedChars

**Purpose** Draw the specified characters inverted (background color) in the draw window.

**Declared In** Window.h

**Prototype** void WinDrawInvertedChars (const Char \*chars,  
Int16 len, Coord x, Coord y)

**Parameters** -> chars Pointer to the characters to draw.  
-> len Length in bytes of the characters to draw.  
-> x x coordinate of the first character to draw (left bound).

-> **y** y coordinate of the first character to draw (top bound).

**Result** Returns nothing.

**Comments** This routine draws the **on** bits and any underline in the background color and the **off** bits in the text color. (Black and white uses copy NOT mode.) This is the standard function for drawing inverted text.

Before calling this function, consider calling [WinSetUnderlineMode](#) and [FntSetFont](#).

**See Also** [WinDrawChar](#), [WinDrawChars](#), [WinDrawTruncChars](#),  
[WinEraseChars](#), [WinInvertChars](#), [WinPaintChar](#),  
[WinPaintChars](#)

# WinDrawLine

**Purpose** Draw a line in the draw window using the current foreground color.

## **Declared In** Window.h

**Prototype** void WinDrawLine (Coord x1, Coord y1, Coord x2, Coord y2)

<b>Parameters</b>	-> x1	x coordinate of line start point.
	-> y1	y coordinate of line start point.
	-> x2	x coordinate of line endpoint.
	-> y2	y coordinate of line endpoint.

**Result** Returns nothing.

**Comments** This function differs from [WinPaintLine](#) in that it always uses winPaint mode (see [WinDrawOperation](#)). WinPaintLine uses the current drawing state transfer mode instead of winPaint.

**See Also** [WinDrawGrayLine](#), [WinEraseLine](#), [WinFillLine](#),  
[WinInvertLine](#), [WinPaintLine](#), [WinPaintLines](#)

## Windows

### Window Functions

---

## WinDrawPixel

**Purpose** Draw a pixel in the draw window using the current foreground color.

**Declared In** Window.h

**Prototype** void WinDrawPixel (Coord x, Coord y)

**Parameters** -> x Pointer to the x coordinate of a pixel.  
-> y Pointer to the y coordinate of a pixel.

**Result** Returns nothing. May display a fatal error message if the draw window's bitmap is compressed.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [WinErasePixel](#), [WinInvertPixel](#), [WinPaintPixel](#), [WinPaintPixels](#)

## WinDrawRectangle

**Purpose** Draw a rectangle in the draw window using the current foreground color.

**Declared In** Window.h

**Prototype** void WinDrawRectangle (const RectangleType \*rP,  
UInt16 cornerDiam)

**Parameters** -> rP Pointer to the rectangle to draw.  
-> cornerDiam Radius of rounded corners. Specify zero for square corners.

**Result** Returns nothing.

<b>Comments</b>	The cornerDiam parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides. This function differs from <a href="#">WinPaintRectangle</a> in that it always uses winPaint mode (see <a href="#">WinDrawOperation</a> ). WinPaintRectangle uses the current drawing state transfer mode instead of winPaint.
-----------------	--

<b>See Also</b>	<a href="#">WinEraseRectangle</a> , <a href="#">WinFillRectangle</a> , <a href="#">WinInvertRectangle</a>
-----------------	--

## **WinDrawRectangleFrame**

<b>Purpose</b>	Draw a rectangular frame in the draw window using the current foreground color.
----------------	---

<b>Declared In</b>	Window.h
--------------------	----------

**Prototype** void WinDrawRectangleFrame (FrameType frame,  
const RectangleType \*rP)

<b>Parameters</b>	-> frame	Type of frame to draw (see <a href="#">FrameType</a> ).
	-> rP	Pointer to the rectangle to frame.

<b>Result</b>	Returns nothing.
---------------	------------------

<b>Comments</b>	The frame is drawn outside the specified rectangle. This function differs from <a href="#">WinPaintRectangleFrame</a> in that it always uses winPaint mode (see <a href="#">WinDrawOperation</a> ). WinPaintRectangleFrame uses the current drawing state transfer mode instead of winPaint.
-----------------	--

<b>See Also</b>	<a href="#">WinDrawGrayRectangleFrame</a> , <a href="#">WinEraseRectangleFrame</a> , <a href="#">WinGetFramesRectangle</a> , <a href="#">WinInvertRectangleFrame</a>
-----------------	---

## Windows

### Window Functions

---

## WinDrawTruncChars

**Purpose** Draw the specified characters in the draw window, truncating the characters to the specified width.

**Declared In** Window.h

**Prototype** void WinDrawTruncChars (const Char \*chars,  
Int16 len, Coord x, Coord y, Coord maxWidth)

**Parameters**

-> chars	Pointer to the characters to draw.
-> len	Length in bytes of the characters to draw.
-> x	x coordinate of the first character to draw (left bound).
-> y	y coordinate of the first character to draw (top bound).
-> maxWidth	Maximum width in pixels of the characters that are to be drawn.

**Result** Returns nothing.

**Comments** Before calling this function, consider calling [WinSetUnderlineMode](#) and [FntSetFont](#).

If drawing all of the specified characters requires more space than maxWidth allows, WinDrawTruncChars draws one less than the number of characters that can fit in maxWidth and then draws an ellipsis (...) in the remaining space. (If the boundary characters are narrower than the ellipsis, more than one character may be dropped to make room.) If maxWidth is narrower than the width of an ellipsis, nothing is drawn.

Use this function to truncate text that may contain multi-byte characters.

**Compatibility** Implemented only if [3.1 New Feature Set](#) is present. To use this function in code intended to be run on earlier versions of Palm OS, link with the PalmOSGlue library and call

`WinGlueDrawTruncChars`. For more information, see [Chapter 75, "PalmOSGlue Library."](#)

**See Also** [WinDrawChar](#), [WinDrawChars](#), [WinDrawInvertedChars](#), [WinEraseChars](#), [WinInvertChars](#), [WinPaintChar](#), [WinPaintChars](#)

## WinEraseChars

**Purpose** Erase the specified characters in the draw window.

**Declared In** `Window.h`

**Prototype** `void WinEraseChars (const Char *chars, Int16 len,  
Coord x, Coord y)`

**Parameters**

-> <code>chars</code>	Pointer to the characters to erase.
-> <code>len</code>	Length in bytes of the characters to erase.
-> <code>x</code>	x coordinate of the first character to erase (left bound).
-> <code>y</code>	y coordinate of the first character to erase (top bound).

**Result** Returns nothing.

**Comments** The `winMask` transfer mode is used to erase the characters. See [WinDrawOperation](#) for more information. This has the effect of erasing only the on bits for the characters rather than the entire text rectangle. This function only works if the foreground color is black and the background color is white.

**See Also** [WinDrawChar](#), [WinDrawChars](#), [WinDrawInvertedChars](#), [WinDrawTruncChars](#), [WinInvertChars](#), [WinPaintChar](#), [WinPaintChars](#)

## Windows

### Window Functions

---

## WinEraseLine

**Purpose** Draw a line in the draw window using the current background color.

**Declared In** Window.h

**Prototype** void WinEraseLine (Coord x1, Coord y1, Coord x2, Coord y2)

**Parameters**

-> x1	x coordinate of line start point.
-> y1	y coordinate of line start point.
-> x2	x coordinate of line endpoint.
-> y2	y coordinate of line endpoint.

**Result** Returns nothing.

**See Also** [WinDrawGrayLine](#), [WinDrawLine](#), [WinFillLine](#),  
[WinInvertLine](#), [WinPaintLine](#), [WinPaintLines](#)

## WinErasePixel

**Purpose** Draw a pixel in the draw window using the current background color.

**Declared In** Window.h

**Prototype** void WinErasePixel (Coord x, Coord y)

**Parameters**

-> x	Pointer to the x coordinate of a pixel.
-> y	Pointer to the y coordinate of a pixel.

**Result** Returns nothing.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [WinDrawPixel](#), [WinInvertPixel](#), [WinPaintPixel](#),  
[WinPaintPixels](#)

## WinEraseRectangle

**Purpose** Draw a rectangle in the draw window using the current background color.

**Declared In** Window.h

**Prototype** void WinEraseRectangle (const RectangleType \*rP,  
                  UInt16 cornerDiam)

**Parameters**

-> rP	Pointer to the rectangle to erase.
-> cornerDiam	Radius of rounded corners. Specify zero for square corners.

**Result** Returns nothing.

**Comments** The cornerDiam parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides.

**See Also** [WinDrawRectangle](#), [WinFillRectangle](#),  
[WinInvertRectangle](#), [WinPaintRectangle](#)

## Windows

### Window Functions

---

## WinEraseRectangleFrame

**Purpose** Draw a rectangular frame in the draw window using the current background color.

**Declared In** Window.h

**Prototype** void WinEraseRectangleFrame (FrameType frame,  
const RectangleType \*rP)

**Parameters** -> frame Type of frame to draw (see [FrameType](#)).  
-> rP Pointer to the rectangle to frame.

**Result** Returns nothing.

**See Also** [WinDrawGrayRectangleFrame](#), [WinDrawRectangleFrame](#),  
[WinGetFramesRectangle](#), [WinInvertRectangleFrame](#),  
[WinPaintRectangleFrame](#)

## WinEraseWindow

**Purpose** Erase the contents of the draw window.

**Declared In** Window.h

**Prototype** void WinEraseWindow (void)

**Parameters** None.

**Result** Returns nothing.

**Comments** [WinEraseRectangle](#) is used to erase the window. This routine doesn't erase the frame around the draw window. See [WinEraseRectangleFrame](#) and [WinGetWindowFrameRect](#).

## WinFillLine

**Purpose** Fill a line in the draw window with the current pattern.

**Declared In** Window.h

**Prototype** void WinFillLine (Coord x1, Coord y1, Coord x2, Coord y2)

**Parameters**

-> x1	x coordinate of line start point.
-> y1	y coordinate of line start point.
-> x2	x coordinate of line endpoint.
-> y2	y coordinate of line endpoint.

**Result** Returns nothing.

**Comments** You can set the current pattern with [WinSetPattern](#).

**See Also** [WinDrawGrayLine](#), [WinDrawLine](#), [WinEraseLine](#),  
[WinInvertLine](#), [WinPaintLine](#), [WinPaintLines](#)

## WinFillRectangle

**Purpose** Draw a rectangle in the draw window with current pattern.

**Declared In** Window.h

**Prototype** void WinFillRectangle (const RectangleType \*rP, UInt16 cornerDiam)

**Parameters**

-> rP	Pointer to the rectangle to draw.
-> cornerDiam	Radius of rounded corners. Specify zero for square corners.

**Result** Returns nothing.

## Windows

### Window Functions

---

<b>Comments</b>	You can set the current pattern with <a href="#">WinSetPattern</a> .  The cornerDiam parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides.
<b>See Also</b>	<a href="#">WinDrawRectangle</a> , <a href="#">WinEraseRectangle</a> , <a href="#">WinInvertRectangle</a> , <a href="#">WinPaintRectangle</a>

## WinGetActiveWindow

<b>Purpose</b>	Return the window handle of the active window.
<b>Declared In</b>	Window.h
<b>Prototype</b>	WinHandle WinGetActiveWindow (void)
<b>Parameters</b>	None.
<b>Result</b>	Returns the handle of the active window. All user input is directed to the active window.
<b>See Also</b>	<a href="#">WinSetActiveWindow</a> , <a href="#">WinGetDisplayWindow</a> , <a href="#">WinGetFirstWindow</a> , <a href="#">WinGetDrawWindow</a>

## WinGetBitmap

<b>Purpose</b>	Return a pointer to a window's bitmap, which holds the window contents.
<b>Declared In</b>	Window.h
<b>Prototype</b>	BitmapType *WinGetBitmap (WinHandle winHandle)
<b>Parameters</b>	-> winHandle Handle of window from which to get the bitmap.

**Result** Returns a pointer to the bitmap or NULL if winHandle is invalid.

<b>Comments</b>	For onscreen windows, the bitmap returned always represents the whole screen. Thus, the top-left corner of the returned bitmap may not be the top-left corner of the window.
<b>Compatibility</b>	Implemented only if <a href="#">3.5 New Feature Set</a> is present.

## WinGetBounds

<b>Purpose</b>	Return the bounds of the current draw window in display-relative coordinates.				
<b>Declared In</b>	Window.h				
<b>Prototype</b>	<pre>void WinGetBounds (WinHandle winH,                     RectangleType *rP)</pre>				
<b>Parameters</b>	<table><tr><td>-&gt; winH</td><td>Handle to a window.</td></tr><tr><td>&lt;- rP</td><td>Pointer to a rectangle.</td></tr></table>	-> winH	Handle to a window.	<- rP	Pointer to a rectangle.
-> winH	Handle to a window.				
<- rP	Pointer to a rectangle.				
<b>Result</b>	Returns nothing.				
<b>Comments</b>	<p>This function returns in rP the bounds of the window represented by winH. This corresponds to the convention used by <a href="#">WinSetBounds</a>, because it takes a window handle as an argument.</p> <p>Prior to Palm OS 4.0, WinGetBounds returned the bounds of the draw window, and did not take a window handle as an argument. If an application needed to determine the bounds of an arbitrary window, the application would call <a href="#">WinSetDrawWindow</a> to temporarily set the draw window to the desired window, then WinGetBounds would be called to get the bounds of the draw window, then <a href="#">WinSetDrawWindow</a> would be called again to restore the draw window. This is no longer necessary.</p>				
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is <b>not</b> present. As of Palm OS 4.0, applications should use <a href="#">WinGetDrawWindowBounds</a> .				
<b>See Also</b>	<a href="#">WinGetWindowExtent</a> , <a href="#">WinGetDrawWindowBounds</a>				

## Windows

### Window Functions

---

## WinGetClip

**Purpose** Return the clipping rectangle of the draw window.

**Declared In** Window.h

**Prototype** void WinGetClip (RectangleType \*rP)

**Parameters** <- rP Pointer to a structure to hold the clipping bounds.

**Result** Returns nothing.

**See Also** [WinSetClip](#)



---

## WinGetCoordinateSystem

**Purpose** Get the coordinate system

**Declared In** Window.h

**Prototype** UInt16 WinGetCoordinateSystem (void)

**Parameters** None.

**Result** Returns a value representing the current coordinate system. See “[Window Coordinate System Constants](#)” on page 1163 for the values that this function can return.

**Comments** Use this function to determine the active coordinate system. Armed with this information, an application can properly initialize graphic primitive coordinates and dimensions, or can modify the coordinate system with [WinSetCoordinateSystem](#).

**Compatibility** Implemented only if the [High-Density Display Feature Set](#) is

present.

**See Also** [WinSetCoordinateSystem](#)

## WinGetDisplayExtent

**Purpose** Return the width and height of the display (the screen).

**Declared In** Window.h

**Prototype** void WinGetDisplayExtent (Coord \*extentX,  
Coord \*extentY)

**Parameters** <- extentX Pointer to the width of the display in pixels.  
<- extentY Pointer to the height of the display in pixels.

**Result** Returns nothing.

## WinGetDisplayWindow

**Purpose** Return the window handle of the display (screen) window.

**Declared In** Window.h

**Prototype** WinHandle WinGetDisplayWindow (void)

**Parameters** None.

**Result** Returns the handle of display window.

**Comments** The display window is created by the system at start-up; it has the same size as the Palm OS drawable area of the physical display (screen).

**See Also** [WinGetDisplayExtent](#), [WinGetActiveWindow](#),  
[WinGetDrawWindow](#)

## Windows

### Window Functions

---

## WinGetDrawWindow

**Purpose** Return the window handle of the current draw window.

**Declared In** Window.h

**Prototype** WinHandle WinGetDrawWindow (void)

**Parameters** None.

**Result** Returns handle of draw window.

**See Also** [WinGetDisplayWindow](#), [WinGetActiveWindow](#),  
[WinSetDrawWindow](#)

## WinGetDrawWindowBounds

**Purpose** Return the bounds of the draw window.

**Declared In** Window.h

**Prototype** void WinGetDrawWindowBounds (RectangleType \*rP)

**Parameters** <- rP Pointer to the window bounds.

**Result** Returns nothing.

**Comments** A pointer to the bounds of the draw window is returned. This function is equivalent to [WinGetBounds](#) that was in Palm OS prior to 4.0.

**See Also** [WinGetBounds](#)

## WinGetFirstWindow

**Purpose** Return a pointer to the first window in the linked list of windows.

**Declared In** Window.h

**Prototype** WinHandle WinGetFirstWindow (void)

**Parameters** None.

**Result** Returns handle of first window.

**Comments** This function is usually used by the system only.

**See Also** [WinGetActiveWindow](#)

## WinGetFramesRectangle

**Purpose** Return the rectangle that includes a rectangle together with the specified frame around it.

**Declared In** Window.h

**Prototype** void WinGetFramesRectangle (FrameType frame,  
const RectangleType \*rP,  
RectangleType \*obscuredRect)

**Parameters** -> frame Type of rectangle frame (see [FrameType](#)).

-> rP Pointer to the rectangle to frame.

<- obscuredRect Pointer to the rectangle that includes both the specified rectangle and its frame.

**Result** Returns nothing.

## Windows

### Window Functions

---

**Comments** Frames are always drawn around (outside) a rectangle.

**See Also** [WinGetWindowFrameRect](#), [WinGetBounds](#)

## WinGetPattern

**Purpose** Return the current fill pattern.

**Declared In** Window.h

**Prototype** void WinGetPattern (CustomPatternType \*patternP)

**Parameters** <- patternP      Buffer where the current pattern is returned  
(see [CustomPatternType](#)).

**Result** Returns nothing.

**Comments** The fill pattern is used by [WinFillLine](#) and [WinFillRectangle](#).

This function returns the value of patternData in the current drawing state. (See [DrawStateType](#).) The patternData field is only set if the pattern field is customPattern. Therefore, it's a good idea to use [WinGetPatternType](#) instead of this function on systems that support [WinGetPatternType](#).

**See Also** [WinSetPattern](#)

## WinGetPatternType

<b>Purpose</b>	Return the current pattern type.
<b>Declared In</b>	Window.h
<b>Prototype</b>	PatternType WinGetPatternType (void)
<b>Parameters</b>	None.
<b>Result</b>	Returns the current draw window pattern type (see <a href="#">PatternType</a> ). If the return value is customPattern, you can retrieve the pattern with <a href="#">WinGetPattern</a> .
<b>Comments</b>	The fill pattern is used by <a href="#">WinFillLine</a> and <a href="#">WinFillRectangle</a> .
<b>Compatibility</b>	Implemented only if <a href="#">3.5 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">WinSetPatternType</a>

## WinGetPixel

<b>Purpose</b>	Return the color value of a pixel in the current draw window.
<b>Declared In</b>	Window.h
<b>Prototype</b>	IndexedColorType WinGetPixel (Coord x, Coord y)
<b>Parameters</b>	-> x                            Pointer to the x coordinate of a pixel. -> y                            Pointer to the y coordinate of a pixel.
<b>Result</b>	Returns the indexed color value of the pixel. See <a href="#">IndexedColorType</a> . A return value of 0 means either that the coordinates do not lie in the current draw window or that they do and the color of that pixel is index 0 (typically white).

## Windows

### Window Functions

---

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [WinIndexToRGB](#)

## WinGetPixelRGB

**Purpose** Return the RGB color values of a pixel in the current draw window.

**Declared In** Window.h

**Prototype** Err WinGetPixelRGB (Coord x, Coord y,  
RGBColorType \*rgbP)

**Parameters** -> x Pointer to the x coordinate of a pixel.  
-> y Pointer to the y coordinate of a pixel.  
<- rgbP RGB color components of the pixel.

**Result** Returns errNone or sysErrParamErr. sysErrParamErr is returned when the x or y arguments are < 0 or when they are outside the bounds of the draw window.

**Comments** The RGB color values of the pixel are returned as an [RGBColorType](#). This function can be used with both indexed or direct color modes. A return value of sysErrParamErr means that the coordinates do not lie in the current draw window.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.



**New**

## WinGetSupportedDensity

**Purpose** Enumerate the various display densities supported by the blitter.

**Declared In** Window.h

**Prototype** Err WinGetSupportedDensity (UInt16 \*densityP)

**Parameters** <-> densityP Pointer to a supported density value. Set this value to zero before calling this function for the first time. Subsequent calls cause this value to be set to one of the display densities supported by the handheld.

**Result** Returns errNone unless the value you supply in \*densityP isn't a supported density and isn't zero, in which case this function returns SysErrParamErr.

**Comments** Initialize \*densityP to zero before your application calls this function for the first time. Repeated calls to WinGetSupportedDensity will cause the value pointed to by densityP to change; these values represent the supported display densities, in order from low to high density. After the last supported density value, this function sets \*densityP back to zero.

---

**NOTE:** The densities reported by this function are those that are supported by the blitter. These densities are not necessarily supported by the underlying hardware. A handheld with a low-density screen that is able to scale high-density bitmaps will report that it can handle both high and low density bitmaps. Use [WinScreenGetAttribute](#) to determine the density of the handheld's screen.

---

Density values are defined in Bitmap.h; see the [DensityType](#) enum. Only those values supported by a given handheld will be returned by WinGetSupportedDensity. For example, on a

## Windows

### Window Functions

---

handheld with a double-density display this function returns kDensityLow, followed by kDensityDouble, followed by 0. For each supported density, the inverse scaling factor is supported. In this example, the blitter supports pixel-doubling low-density data for a double-density destination, and the blitter supports pixel-halving high-density data for a low-density destination.

The value pointed to by densityP should only be zero or one of the density values supported by the handheld. If it has any other value when you call WinGetSupportedDensity, this function will simply return sysErrParamErr.

<b>Compatibility</b>	Implemented only if the <a href="#">High-Density Display Feature Set</a> is present.
----------------------	--

## WinGetWindowExtent

**Purpose** Return the width and height of the current draw window.

**Declared In** Window.h

**Prototype** void WinGetWindowExtent (Coord \*extentX,  
Coord \*extentY)

**Parameters** <- extentX Pointer to the width in pixels of the draw window.  
<- extentY Pointer to the height in pixels of the draw window.

**Result** Returns nothing.

**See Also** [WinGetBounds](#), [WinGetWindowFrameRect](#),

## WinGetWindowFrameRect

**Purpose** Return a rectangle, in display-relative coordinates, that defines the size and location of a window and its frame.

**Declared In** Window.h

**Prototype** void WinGetWindowFrameRect (WinHandle winHandle,  
RectangleType \*r)

**Parameters** -> winHandle Handle of window whose coordinates are  
desired.  
<- r Pointer to the coordinates of the window.

**Result** Returns nothing.

**See Also** [WinGetBounds](#)

## WinIndexToRGB

**Purpose** Convert an index in the currently active color table to an RGB value.

**Declared In** Window.h

**Prototype** void WinIndexToRGB (IndexedColorType i,  
RGBColorType \*rgbP)

**Parameters** -> i A color index value. See [IndexedColorType](#).  
<- rgbP Pointer to an RGB color value corresponding to  
the index value i. See [RGBColorType](#).

**Result** Returns nothing.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [WinRGBToIndex](#)

## Windows

### Window Functions

---

## WinInvertChars

**Purpose** Invert the specified characters in the draw window.

**Declared In** Window.h

**Prototype** void WinInvertChars (const Char \*chars,  
Int16 len, Coord x, Coord y)

**Parameters**

-> chars	Pointer to the characters to invert.
-> len	Length in bytes of the characters to invert.
-> x	x coordinate of the first character to invert (left bound).
-> y	y coordinate of the first character to invert (top bound).

**Result** Returns nothing.

**Comments** This function applies the winInvert operation of [WinDrawOperation](#) to the characters in the draw window.

To perform color inverting, use [WinSetTextColor](#) and [WinSetBackColor](#) to choose the desired colors, and call [WinPaintChar](#).

**See Also** [WinDrawChar](#), [WinDrawChars](#), [WinDrawInvertedChars](#),  
[WinDrawTruncChars](#), [WinEraseChars](#), [WinPaintChar](#),  
[WinPaintChars](#)

## **WinInvertLine**

**Purpose** Invert a line in the draw window (using the [WinDrawOperation](#) `winInvert`).

**Declared In** Window.h

**Prototype** `void WinInvertLine (Coord x1, Coord y1, Coord x2, Coord y2)`

**Parameters**

-> x1	x coordinate of line start point.
-> y1	y coordinate of line start point.
-> x2	x coordinate of line endpoint.
-> y2	y coordinate of line endpoint.

**Result** Returns nothing.

**See Also** [WinDrawGrayLine](#), [WinDrawLine](#), [WinEraseLine](#),  
[WinFillLine](#), [WinPaintLine](#), [WinPaintLines](#)

## **WinInvertPixel**

**Purpose** Invert a pixel in the draw window (using the [WinDrawOperation](#) `winInvert`).

**Declared In** Window.h

**Prototype** `void WinInvertPixel (Coord x, Coord y)`

**Parameters**

-> x	Pointer to the x coordinate of a pixel.
-> y	Pointer to the y coordinate of a pixel.

**Result** Returns nothing.

## Windows

### Window Functions

---

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [WinDrawPixel](#), [WinErasePixel](#), [WinPaintPixel](#),  
[WinPaintPixels](#)

## WinInvertRectangle

**Purpose** Invert a rectangle in the draw window (using the [WinDrawOperation](#) `winInvert`).

**Declared In** `Window.h`

**Prototype** `void WinInvertRectangle (const RectangleType *rP,  
                  UInt16 cornerDiam)`

**Parameters** `-> rP` Pointer to the rectangle to invert.

`-> cornerDiam` Radius of rounded corners. Specify zero for square corners.

**Result** Returns nothing.

**Comments** The `cornerDiam` parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides.

The operating system itself does not use the inverting routines. Instead, it uses the `winSwap` transfer mode, or it changes the color selection and uses the `WinPaint...` routines.

**See Also** [WinDrawRectangle](#), [WinEraseRectangle](#),  
[WinFillRectangle](#), [WinPaintRectangle](#)

## WinInvertRectangleFrame

**Purpose** Invert a rectangular frame in the draw window (using the [WinDrawOperation](#) `winInvert`).

**Declared In** Window.h

**Prototype** void WinInvertRectangleFrame (FrameType frame,  
const RectangleType \*rP)

**Parameters** -> frame Type of frame to draw (see [FrameType](#)).  
-> rP Pointer to the rectangle to frame.

**Result** Returns nothing.

**See Also** [WinDrawGrayRectangleFrame](#), [WinDrawRectangleFrame](#),  
[WinEraseRectangleFrame](#), [WinGetFramesRectangle](#),  
[WinPaintRectangleFrame](#)

## WinModal

**Purpose** Return true if the specified window is modal.

**Declared In** Window.h

**Prototype** Boolean WinModal (WinHandle winHandle)

**Parameters** -> winHandle Handle of a window.

**Result** Returns true if the window is modal, otherwise false.

**Comments** A window is modal if it cannot lose the focus.

**See Also** [FrmAlert](#), [FrmCustomAlert](#), [FrmDoDialog](#)

## WinPaintBitmap

<b>Purpose</b>	Draw a bitmap in the current draw window at the specified coordinates with the current draw mode.						
<b>Declared In</b>	Window.h						
<b>Prototype</b>	<code>void WinPaintBitmap (BitmapType *bitmapP, Coord x, Coord y)</code>						
<b>Parameters</b>	<table><tr><td>-&gt; bitmapP</td><td>Pointer to a bitmap.</td></tr><tr><td>-&gt; x</td><td>The x coordinate of the top-left corner.</td></tr><tr><td>-&gt; y</td><td>The y coordinate of the top-left corner.</td></tr></table>	-> bitmapP	Pointer to a bitmap.	-> x	The x coordinate of the top-left corner.	-> y	The y coordinate of the top-left corner.
-> bitmapP	Pointer to a bitmap.						
-> x	The x coordinate of the top-left corner.						
-> y	The y coordinate of the top-left corner.						
<b>Result</b>	Returns nothing.						
<b>Comments</b>	<p>If the bitmap has multiple depths (is a bitmap family), the closest match less than or equal to the current draw window depth is used. If such a bitmap does not exist, the bitmap with the closest match greater than the draw window depth is used.</p> <p>Using <code>WinPaintBitmap</code> is now recommended instead of the previous practice of rendering bitmaps into an offscreen window and then using <a href="#">WinCopyRectangle</a> to draw them on screen.</p> <p>The current draw mode is set by <a href="#">WinSetDrawMode</a>.</p> <p>If the bitmap has its own color table, color conversion to the draw window color table will be applied (on OS 3.5 or later). This color conversion is slow and not recommended. Instead of including a color table in the bitmap, consider using <a href="#">WinPalette</a> to change the system color table, draw the bitmap, and then change the system color table back when the bitmap is no longer visible.</p>						
<b>Compatibility</b>	Implemented only if <a href="#">3.5 New Feature Set</a> is present.						
<b>See Also</b>	<a href="#">WinDrawBitmap</a> , <a href="#">WinEraseRectangle</a> , <a href="#">WinPaintTiledBitmap</a>						

## WinPaintChar

**Purpose** Draw a character in the draw window using the current drawing state.

**Declared In** Window.h

**Prototype** void WinPaintChar (WChar theChar, Coord x,  
Coord y)

**Parameters**

-> theChar	The character to draw. This may be either a single-byte character or a multi-byte character.
-> x	x coordinate of the location where the character is to be drawn (left bound).
-> y	y coordinate of the location where the character is to be drawn (top bound).

**Result** Returns nothing.

**See Also** WinPaintChar draws the **on** bits in the text color and the **off** bits in the background color, with underlines (if any) drawn in the foreground color using the current drawing mode.

This function uses the current drawing state, which is stored in a [DrawStateType](#) structure. See the description of that structure to learn the functions you can call to set the drawing state to the values you want.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [WinDrawChar](#), [WinDrawChars](#), [WinDrawInvertedChars](#),  
[WinDrawTruncChars](#), [WinEraseChars](#), [WinInvertChars](#),  
[WinPaintChars](#)

## Windows

### Window Functions

---

## WinPaintChars

**Purpose** Draw the specified characters in the draw window with the current draw state.

**Declared In** Window.h

**Prototype** void WinPaintChars (const Char \*chars, Int16 len,  
Coord x, Coord y)

**Parameters**

-> chars	Pointer to the characters to draw.
-> len	Length in bytes of the characters to draw.
-> x	x coordinate of the first character to draw (left bound).
-> y	y coordinate of the first character to draw (top bound).

**Result** Returns nothing.

**Comments** WinPaintChars draws the **on** bits in the text color and the **off** bits in the background color, with underlines (if any) drawn in the foreground color using the current drawing mode.

This function uses the current drawing state, which is stored in a [DrawStateType](#) structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

Before calling this function, consider calling [WinSetUnderlineMode](#) and [FntSetFont](#).

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [WinDrawChar](#), [WinDrawChars](#), [WinDrawInvertedChars](#),  
[WinDrawTruncChars](#), [WinEraseChars](#), [WinInvertChars](#),  
[WinPaintChar](#)

## WinPaintLine

**Purpose** Draw a line in the draw window using the current drawing state.

**Declared In** Window.h

**Prototype** void WinPaintLine (Coord x1, Coord y1, Coord x2, Coord y2)

**Parameters**

-> x1	x coordinate of line beginning point.
-> y1	y coordinate of line beginning point.
-> x2	x coordinate of line endpoint.
-> y2	y coordinate of line endpoint.

**Result** Returns nothing.

**Comments** This function uses the current drawing state, which is stored in a [DrawStateType](#) structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [WinDrawLine](#), [WinDrawGrayLine](#), [WinEraseLine](#), [WinFillLine](#), [WinInvertLine](#), [WinPaintLines](#)

## WinPaintLines

**Purpose** Draw several lines in the draw window using the current drawing state.

**Declared In** Window.h

**Prototype** void WinPaintLines (UInt16 numLines, WinLineType lines[] )

**Parameters**

-> numLines	Number of lines to paint.
-------------	---------------------------

## Windows

### Window Functions

---

-> lines      Array of lines. See [WinLineType](#).

**Result** Returns nothing.

**Comments** This function uses the current drawing state, which is stored in a [DrawStateType](#) structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [WinDrawLine](#), [WinDrawGrayLine](#), [WinEraseLine](#),  
[WinFillLine](#), [WinInvertLine](#), [WinPaintLine](#)

## WinPaintPixel

**Purpose** Render a pixel in the draw window using the current drawing state.

**Declared In** Window.h

**Prototype** void WinPaintPixel (Coord x, Coord y)

**Parameters** -> x      Pointer to the x coordinate of a pixel.  
-> y      Pointer to the y coordinate of a pixel.

**Result** Returns nothing.

**Comments** This function uses the current drawing state, which is stored in a [DrawStateType](#) structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [WinDrawPixel](#), [WinErasePixel](#), [WinInvertPixel](#),  
[WinPaintPixels](#)

## WinPaintPixels

<b>Purpose</b>	Render several pixels in the draw window using the current drawing state.				
<b>Declared In</b>	Window.h				
<b>Prototype</b>	<code>void WinPaintPixels (UInt16 numPoints, PointType pts[])</code>				
<b>Parameters</b>	<table><tr><td>-&gt; numPoints</td><td>Number of pixels to paint.</td></tr><tr><td>-&gt; pts</td><td>Array of pixels.</td></tr></table>	-> numPoints	Number of pixels to paint.	-> pts	Array of pixels.
-> numPoints	Number of pixels to paint.				
-> pts	Array of pixels.				
<b>Result</b>	Returns nothing.				
<b>Comments</b>	This function uses the current drawing state, which is stored in a <a href="#">DrawStateType</a> structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.				
<b>Compatibility</b>	Implemented only if <a href="#">3.5 New Feature Set</a> is present.				
<b>See Also</b>	<a href="#">WinDrawPixel</a> , <a href="#">WinErasePixel</a> , <a href="#">WinInvertPixel</a> , <a href="#">WinPaintPixel</a>				

## WinPaintRectangle

<b>Purpose</b>	Draw a rectangle in the draw window using the current drawing state.		
<b>Declared In</b>	Window.h		
<b>Prototype</b>	<code>void WinPaintRectangle (const RectangleType *rP, UInt16 cornerDiam)</code>		
<b>Parameters</b>	<table><tr><td>-&gt; rP</td><td>Pointer to the rectangle to draw.</td></tr></table>	-> rP	Pointer to the rectangle to draw.
-> rP	Pointer to the rectangle to draw.		

## Windows

### Window Functions

---

-> cornerDiam    Radius of rounded corners. Specify zero for square corners.

**Result**    Returns nothing.

**Comments**    The cornerDiam parameter specifies the radius of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides. This function uses the current drawing state, which is stored in a [DrawStateType](#) structure. See the description of that structure to learn the functions you can call to set the drawing state to the state you want.

**Compatibility**    Implemented only if [3.5 New Feature Set](#) is present.

**See Also**    [WinDrawRectangle](#), [WinEraseRectangle](#),  
[WinFillRectangle](#), [WinInvertRectangle](#)

## WinPaintRectangleFrame

**Purpose**    Draw a rectangular frame in the draw window using the current drawing state.

**Declared In**    Window.h

**Prototype**    void WinPaintRectangleFrame (FrameType frame,  
                  const RectangleType \*rP)

**Parameters**    -> frame              Type of frame to draw (see [FrameType](#)).  
                  -> rP                 Pointer to the rectangle to frame.

**Result**    Returns nothing.

**Comments**    The frame is drawn outside the specified rectangle. This function uses the current drawing state, which is stored in a [DrawStateType](#) structure. See the description of that structure to

learn the functions you can call to set the drawing state to the state you want.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [WinDrawGrayRectangleFrame](#), [WinDrawRectangleFrame](#),  
[WinEraseRectangleFrame](#), [WinGetFramesRectangle](#),  
[WinInvertRectangleFrame](#),  
[WinPaintRoundedRectangleFrame](#)



## WinPaintRoundedRectangleFrame

**Purpose** Draw a rectangular frame with rounded corners in the draw window using the current drawing state.

**Declared In** Window.h

**Prototype**

```
void WinPaintRoundedRectangleFrame  
(const RectangleType *rP, Coord width,  
Coord cornerRadius, Coord shadowWidth)
```

**Parameters**

-> rP	Pointer to the rectangle to frame.
-> width	The width of the frame, interpreted using the active coordinate system.
-> cornerRadius	The radius of the rectangle's rounded corners, interpreted using the active coordinate system.
-> shadowWidth	The shadow offset, interpreted using the active coordinate system.

**Result** Returns nothing.

**Comments** This function allows you to draw a rectangle with a frame width and corner radius specified in the active coordinate system. It is necessary because [WinPaintRectangleFrame](#) doesn't allow you to draw rounded rectangles with a frame width greater than 2. Note that because there isn't a function that parallels either

## Windows

### Window Functions

---

[WinDrawRectangleFrame](#), [WinEraseRectangleFrame](#), or [WinInvertRectangleFrame](#), you must set the drawing mode and colors as appropriate and use [WinPaintRoundedRectangleFrame](#) to achieve the desired effect.

<b>Compatibility</b>	Implemented only if the <a href="#">High-Density Display Feature Set</a> is present.
----------------------	--



## WinPaintTiledBitmap

<b>Purpose</b>	Fill a rectangle with a pattern defined by a bitmap.
----------------	--

<b>Declared In</b>	Window.h
--------------------	----------

<b>Prototype</b>	void WinPaintTiledBitmap (BitmapType *bitmapP, RectangleType *rectP)
------------------	---

<b>Parameters</b>	-> bitmapP      Pointer to the bitmap that contains the desired pattern. -> rectP        Pointer to the rectangle that is to be filled.
-------------------	--

<b>Result</b>	Returns nothing. On a debug ROM, if either bitmapP or rectP are NULL, an error is displayed.
---------------	--

<b>Comments</b>	This function makes it possible for an application to define a pattern that is larger than the standard 8 by 8 custom pattern, and to define high-density custom patterns.
-----------------	--

The pattern is scaled by the blitter using the density of bitmapP and the density of the screen bitmap. bitmapP can be a bitmap family; if it is, the Window Manager selects a bitmap using the same algorithm used by [WinPaintBitmap](#). As with other patterns, the tiled pattern is anchored to the window's origin.

If bitmapP does not match the depth or density of the destination bitmap, the blitter converts the bitmap using a temporary buffer.

Note that if there isn't enough heap space for the temporary buffer, WinPaintTiledBitmap will be slow.

<b>Compatibility</b>	Implemented only if the <a href="#">High-Density Display Feature Set</a> is present.
----------------------	--

## WinPalette

**Purpose** Set or retrieve the palette for the draw window.

**Declared In** Window.h

**Prototype** Err WinPalette (UInt8 operation,  
Int16 startIndex, UInt16 paletteEntries,  
RGBColorType \*tableP)

<b>Parameters</b>	-> operation	Specify one of the following values:  winPaletteGet Retrieve the palette. Entries are read from the palette beginning at startIndex and placed into tableP beginning at index 0.  winPaletteSet Set the palette. Entries from tableP (beginning at index 0) are set into the palette beginning at startIndex in the palette.  winPaletteSetToDefault Set the palette to the default system palette.
-------------------	--------------	--

## Windows

### Window Functions

---

-> startIndex Identifies where in the palette to start reading or writing. Specify WinUseTableIndexes to indicate that the entries are not to be set or read sequentially; instead, the index value in each RGBColorType entry in tableP determines which slot in the palette is to be set or read. You can use this technique to get or set several discontiguous palette entries with a single function call.

-> paletteEntries Number of palette entries to get or set.

<-> tableP A pointer to a buffer of [RGBColorType](#) entries that is either read from or written to, depending on the operation parameter; the table entries from 0 to paletteEntries – 1 are affected by this routine.

**Result** Returns one of the following values:

errNone Success.

winErrPalette The current draw window does not have a color table, a set operation has overflowed the color table, or one of the entries in tableP has an invalid index value

sysErrParamErr The startIndex value is invalid.

**Comments** Here are some examples of how this routine works:

- If startIndex is 0 and paletteEntries is 10, the first 10 elements of the palette will be set from tableP or will be copied into tableP.
- If startIndex is 10 and paletteEntries is 5, then entries 10, 11, 12, 13, and 14 in the palette will be set from or copied to elements 0, 1, 2, 3, and 4 in tableP.
- If startIndex is WinUseTableIndexes and paletteEntries is 1, then the index value in the RGBColorType of element 0 of tableP will be read from or copied to tableP; in this case, the index field of the RGBColorType will not change.

During a set operation, this function broadcasts the [sysNotifyDisplayChangeEvent](#) to notify any interested observer that the color palette has changed.

One use for this function is if you need to display a bitmap that uses a color table other than the one in use by the system. You can attach a custom color table to a bitmap, and if you do, the bitmap is drawn using that color table. However, this is a performance drain. As an optimization, you can use `WinPalette` to change the system color table to match that used by the bitmap, display the bitmap, and use `WinPalette` to reset the color table when the bitmap is no longer visible.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

## **WinPopDrawState**

**Purpose** Restore the draw state values to the last saved set on the stack.

**Declared In** `Window.h`

**Prototype** `void WinPopDrawState (void)`

**Parameters** None.

**Result** Returns nothing.

**Comments** Use this routine to restore the draw state saved by the previous call to [WinPushDrawState](#).

After you call this function, the current draw window's `drawStateP` field points to the restored drawing state.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

## Windows

### Window Functions

---

## WinPushDrawState

**Purpose** Save the current draw state values onto the draw state stack.

**Declared In** Window.h

**Prototype** void WinPushDrawState (void)

**Parameters** None.

**Result** Returns nothing.

**Comments** Use this routine to save the current draw state before making changes to it using the functions listed in the [DrawStateType](#) structure's description. Call [WinPopDrawState](#) to restore the saved settings.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

## WinResetClip

**Purpose** Reset the clipping rectangle of the draw window to the portion of the draw window that is within the bounds of the display.

**Declared In** Window.h

**Prototype** void WinResetClip (void)

**Parameters** None.

**Result** Returns nothing.

**See Also** [WinSetClip](#)

## WinRestoreBits

**Purpose** Copy the contents of the specified window to the draw window and delete the passed window.

**Declared In** Window.h

**Prototype** void WinRestoreBits (WinHandle winHandle,  
Coord destX, Coord destY)

**Parameters** -> winHandle Handle of window to copy and delete.  
-> destX x coordinate in the draw window to copy to.  
-> destY y coordinate in the draw window to copy to.

**Result** Returns nothing.

**Comments** This routine is generally used to restore a region of the display that was saved with [WinSaveBits](#).

**See Also** [WinSaveBits](#)

## WinRGBToIndex

**Purpose** Convert an RGB value to the index of the closest color in the currently active color lookup table (CLUT).

**Declared In** Window.h

**Prototype** IndexedColorType WinRGBToIndex  
(const RGBColorType \*rgbP)

**Parameters** -> rgbP Pointer to an RGB color value.

**Result** Returns the index of the closest matching color in the CLUT.

**Comments** Palm OS 3.5 supports a maximum of 256 colors. The number of possible RGB colors greatly exceeds this amount. For this reason, an

## Windows

### Window Functions

---

exact match may not be available for `rgbP`. If there is no exact RGB match, then a luminance best-fit is used if the color lookup table is entirely gray scale (red, green, and blue values for each entry are identical), or a shortest-distance fit in RGB space is used if the palette contains colors. RGB shortest distance may not always produce the actual closest perceptible color, but it's relatively fast and works for the system palette.

`WinRGBToIndex` uses the draw window's color table to return the appropriate color table index. If the draw window does not have a color table, the default color table of the current screen is used.

If the draw window does not have a color table, and if the depth of the draw window and the depth of the screen are different, this function will return an inappropriate index. If this situation exists, the application should either define a color table for the draw window, or use `WinScreenMode` to set the screen depth to the same depth as the draw window before calling `WinRGBToIndex`.

---

**NOTE:** The bitmap data will not be blitted properly if the depth of the screen is changed using `WinScreenMode` and the new window uses a bitmap that does not define the bitmap's color table. See `WinScreenMode` for information on how to work around this limitation.

---

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [WinIndexToRGB](#), [WinScreenMode](#)

## WinSaveBits

**Purpose** Create an offscreen window and copy the specified region from the draw window to the offscreen window.

**Declared In** Window.h

**Prototype**

```
WinHandle WinSaveBits  
(const RectangleType *source, UInt16 *error)
```

**Parameters**

-> source	Pointer to the bounds of the region to save, relative to the display.
<- error	Pointer to any error encountered by this function.

**Result** Returns the handle of the window containing the saved image, or zero if an error occurred.

**Comments** The offscreen window is the same size as the region to copy.

This function tries to copy the window's bitmap using compressed format if possible. It may display a fatal error message if an error occurs when it tries to shrink the pointer for the compressed bits.

**See Also** [WinRestoreBits](#)

**New**

## WinScaleCoord

**Purpose** Convert a single coordinate from the standard coordinate system to the active coordinate system.

**Declared In** Window.h

**Prototype** Coord WinScaleCoord (Coord coord,  
Boolean ceiling)

**Parameters**

-> coord	A coordinate in the standard coordinate system.
-> ceiling	Pass true to round up, false to truncate the fractional part when scaling.

**Result** Returns the coordinate scaled to the active coordinate system.

**Comments** This function converts a coordinate by multiplying it by the coordinate scaling factor, and then truncating or rounding the result to an integer value depending on the value of ceiling.

If the active coordinate system is kCoordinatesStandard, the returned coordinate is equal to the supplied coordinate.

**Compatibility** Implemented only if the [High-Density Display Feature Set](#) is present.

**See Also** [WinScalePoint](#), [WinScaleRectangle](#), [WinUnscaleCoord](#)



**New**

## WinScalePoint

**Purpose** Convert a point from the standard coordinate system to the active coordinate system.

**Declared In** Window.h

**Prototype** void WinScalePoint (PointType \*pointP,  
Boolean ceiling)

**Parameters** <-> pointP Pointer to a PointType structure that, before the call, should contain a point's standard coordinate system coordinates. After this function is called the PointType structure contains the coordinates of the point scaled to the active coordinate system.

-> ceiling Pass true to round up, false to truncate the fractional part when scaling.

**Result** Returns nothing. The coordinates of the point indicated by pointP are converted to the active coordinate system.

**Comments** This function converts a point by multiplying its x and y coordinates by the coordinate scaling factor and then truncating or rounding the results depending on the value of ceiling. If the active coordinate system is kCoordinatesStandard, pointP is not changed by this function.

**Compatibility** Implemented only if the [High-Density Display Feature Set](#) is present.

**See Also** [WinScaleCoord](#), [WinScaleRectangle](#), [WinUnscalePoint](#)

**New**

## WinScaleRectangle

**Purpose** Convert a rectangle from the standard coordinate system to the active coordinate system.

**Declared In** Window.h

**Prototype** void WinScaleRectangle (RectangleType \*rectP)

**Parameters** <-> rectP Pointer to a RectangleType structure that, before the call, should contain a rectangle's standard coordinate system coordinates. After this function is called the RectangleType structure contains the coordinates of the rectangle scaled to the active coordinate system.

**Result** Returns nothing. The coordinates of the rectangle indicated by rectP are converted to the native coordinate system.

**Comments** This function scales the rectangle's topLeft and extent points by multiplying their x and y coordinates by the coordinate scaling factor. All values are then truncated, but if either topLeft.x or extent.x had a fractional part, extent.x is incremented by 1 (and, similarly, if either topLeft.y or extent.y had a fractional part, extent.y is incremented by 1).

If the active coordinate system is kCoordinatesStandard, rectP is not changed by this function.

You can use this function when your gadget handler draws using a more precise coordinate system than the Form Manager and needs to convert the form-based bounds of the gadget to the high-density bounds used by the gadget's drawing function.

**Compatibility** Implemented only if the [High-Density Display Feature Set](#) is

present.

**See Also** [WinScaleCoord](#), [WinScalePoint](#), [WinUnscaleRectangle](#)



## New **WinScreenGetAttribute**

**Purpose** Get various attributes of the screen.

**Declared In** Window.h

**Prototype** Err WinScreenGetAttribute  
(WinScreenAttrType selector, UInt32 \*attrP)

**Parameters**

-> selector	A value indicating which attribute to return. See the description of <a href="#">WinScreenAttrType</a> in the Comments section, below, for the values you can supply to this parameter.
<- attrP	Pointer to a UInt32 into which the specified attribute value is placed by this function.

**Result** Returns errNone if the function successfully retrieved the specified attribute, or sysErrParamErr if selector doesn't represent a screen attribute.

**Comments** This function returns many of the attributes that can be obtained with [WinScreenMode](#). Unlike WinScreenMode, however, this function can also return the number of bytes used by each row in the screen buffer as well as the number of pixels per inch on the screen's x and y axes.

Unlike WinScreenMode, you cannot set any attributes with this function. Also, you cannot use this function to obtain the "color enabled" attribute. And unlike WinScreenMode, this function always returns the true screen dimensions; WinScreenMode converts the dimensions to the active coordinate system.

## Windows

### Window Functions

---

Applications can use the screen resolution information to make intelligent decisions about how to draw primitives on Palm Powered handhelds with different screen resolutions.

#### **WinScreenAttrType**

This enum defines the selectors that can be used with the [WinScreenGetAttribute](#) function.

```
typedef enum {
    winScreenWidth,
    winScreenHeight,
    winScreenRowBytes,
    winScreenDepth,
    winScreenAllDepths,
    winScreenDensity,
    winScreenPixelFormat,
    winScreenResolutionX,
    winScreenResolutionY
} WinScreenAttrType;
```

#### **Value Descriptions**

winScreenWidth	The width of the screen, in pixels.
winScreenHeight	The height of the screen, in pixels.
winScreenRowBytes	The number of bytes used by each row in the screen buffer.
winScreenDepth	The screen depth.
winScreenAllDepths	All screen depths (in bitmap format).
winScreenDensity	The screen bitmap's density.
winScreenPixelFormat	The <a href="#">PixelFormatType</a> appropriate for the screen.
winScreenResolutionX	The number of pixels per inch along the screen's x axis.
winScreenResolutionY	The number of pixels per inch along the screen's yaxis.

**Compatibility** Implemented only if the [High-Density Display Feature Set](#) is

present.

**See Also** [WinScreenMode](#)

## WinScreenLock

**Purpose** “Lock” the current screen by switching the UI concept of the screen base address to an area that is not reflected on the display.

**Declared In** Window.h

**Prototype** UInt8 \*WinScreenLock (WinLockInitType initMode)

**Parameters** -> initMode      Indicates how to initialize the new screen area.  
Specify one of the following values:  
winLockCopy  
    Copy old screen to new.  
winLockErase  
    Erase new screen to white.  
winLockDontCare  
    Don't do anything

**Result** Returns a pointer to the new screen base address, or NULL if this routine fails.

**Comments** This routine can be used to “freeze” the display while doing lengthy drawing operations to avoid a flickering effect. Call [WinScreenUnlock](#) to unlock the display and cause it to be updated with any changes. The screen must be unlocked as many times as it is locked to actually update the display.  
Because this function copies the screen, using it is a relatively expensive operation.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

## WinScreenMode

**Purpose** Sets or returns display parameters, including display geometry, bit depth, and color support.

**Declared In** Window.h

**Prototype**

```
Err WinScreenMode
    (WinScreenModeOperation operation,
     UInt32 *widthP, UInt32 *heightP, UInt32 *depthP,
     Boolean *enableColorP)
```

**Parameters** The widthP, heightP, depthP, and enableColorP parameters are used in different ways for different operations. See [Comments](#) at the end of this description for details.

-> operation The work this function is to perform, as specified by one of the following selectors:

winScreenModeGet

Return the current settings for the display.

winScreenModeGetDefaults

Return the default settings for the display.

winScreenModeGetSupportedDepths

Return in depthP a hexadecimal value indicating the supported screen depths. The binary representation of this value defines a bitfield in which the value 1 indicates support for a particular display depth. The position representing a particular bit depth corresponds to the value  $2^{(\text{bitDepth}-1)}$ . See the [Example](#) at the end of this function description for more information.

	<code>winScreenModeGetSupportsColor</code> Return true as the value of the enableColorP parameter when color mode can be enabled.
	<code>winScreenModeSet</code> Change display settings to the values specified by the other arguments to the <code>WinScreenMode</code> function.
	<code>winScreenModeSetToDefaults</code> Change display settings to default values.
<-> widthP	Pointer to new/old screen width. For backward compatibility, when operation is <code>winScreenModeGet</code> or <code>winScreenModeGetDefaults</code> , a single-density width is returned, even if the handheld has a double-density display. Use <a href="#"><u>WinScreenGetAttribute</u></a> to retrieve the true hardware dimensions of the display.
<-> heightP	Pointer to new/old screen height. For backward compatibility, when operation is <code>winScreenModeGet</code> or <code>winScreenModeGetDefaults</code> , a single-density height is returned, even if the handheld has a double-density display. Use <a href="#"><u>WinScreenGetAttribute</u></a> to retrieve the true hardware dimensions of the display.
<-> depthP	Pointer to new/old/available screen depth.

## Windows

### Window Functions

---

<-> enableColorP

Pass true to enable color drawing mode. The returned value (when using an operation that returns a value through this parameter) simply indicates whether or not the hardware supports color; its value does not change based on the current screen depth.

**Result** If no error, returns values as specified by the operation argument. Various invalid arguments may cause this function to return a sysErrParamErr result code. In rare cases, a failed allocation can cause this function to return a memErrNotEnoughSpace error.

**Comments** The widthP, heightP, depthP, and enableColorP parameters are used in different ways for different operations. All “get” operations overwrite these values with a result when the function returns. The winScreenModeSet operation changes current display parameters when passed valid argument values that are not NULL pointers. The winScreenModeSetToDefaults operation ignores values passed for all of these parameters.

[Table 54.1](#) summarizes parameter usage for each operation this function performs.

**Table 54.1 Use of parameters to WinScreenMode function**

<b>Operation</b> <code>winScreenMode...</code>	<b>widthP</b>	<b>heightP</b>	<b>depthP</b>	<b>enableColorP</b>
...Get	returned	returned	returned	returned
...GetDefaults	returned	returned	returned	returned
...GetSupportedDepths	ignored	ignored	returned	pass in
...GetSupportsColor	ignored	ignored	pass in	returned
...Set	pass in	pass in	pass in	pass in
...SetToDefaults	ignored	ignored	ignored	ignored

This function ignores NULL pointer arguments to the widthP, heightP, depthP, and enableColorP parameters; thus, you can pass a NULL pointer for any of these values to leave the current value unchanged. Similarly, when getting values, this function does not return a value for any NULL pointer argument.

If you change the display depth, it is recommended that you restore it to its previous state when your application closes, even though the system sets display parameters back to their default values when launching an application.

Note that none of the other operations interprets the depth parameter the same way that `winScreenModeGetSupportedDepths` does. For example, to set the display depth to 8-bit mode, you use 8 (decimal) for the display depth, not 0x80 (128 decimal).

When a window is created, and if the window's associated bitmap does not have its own color table, the window will use the system's default color translation tables when a blitting operation occurs to that window. When the system's bit depth changes, the system's default color translation tables are recalculated based on the new screen depth. When the blit occurs at the new screen depth to the offscreen window, the color translation tables are out of sync.

To workaround this system limitation, developers should either:

- allocate offscreen windows after changing the depth, or

## Windows

### Window Functions

---

- use [WinCreateBitmapWindow](#) so that it uses a bitmap with a defined color table.

The latter workaround causes the system to perform color matching when blitting, so the first workaround may be preferred.

#### Compatibility

Implemented only if [3.5 New Feature Set](#) is present. In OS versions prior to 3.5, this function is called `ScrDisplayMode`. The prototype for `ScrDisplayMode` is similar to `WinScreenMode`:

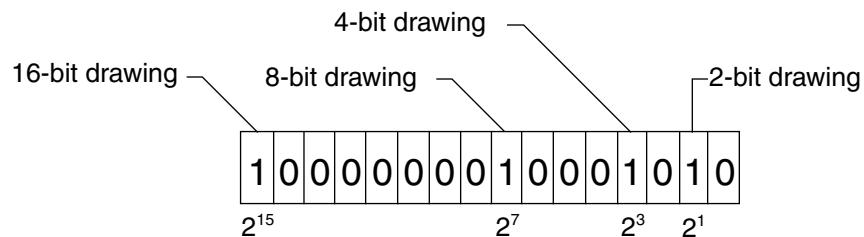
```
Err ScrDisplayMode (
    ScrDisplayModeOperation operation,
    DWordPtr widthP, DWordPtr heightP,
    DWordPtr depthP, BooleanPtr enableColorP)
```

The only other difference between `ScrDisplayMode` and `WinScreenMode` is that the `ScrDisplayModeOperation` constants begin with the prefix `scrDisplayMode` rather than `winScreenMode`.

#### Example

Here are some additional examples of return values provided by the `winScreenModeGetSupportedDepths` mode of the `WinScreenMode` function.

This function indicates support for 4-bit drawing by returning a value of `0x08`, or  $2^3$ , which corresponds to a binary value of 1000. Support for bit depths of 2 and 1 is indicated by a return value of `0x03`. Support for bit depths of 4, 2, and 1 is indicated by `0x0B`, which is a binary value of 1011. Support for bit depths of 16, 8, 4 and 2 is indicated by `0x808A`. The figure immediately following depicts this final example graphically.



#### See Also

[WinScreenGetAttribute](#)

## WinScreenUnlock

<b>Purpose</b>	Unlock the screen and update the display.
<b>Declared In</b>	Window.h
<b>Prototype</b>	void WinScreenUnlock (void)
<b>Parameters</b>	None.
<b>Result</b>	Returns nothing.
<b>Comments</b>	The screen must be unlocked as many times as it is locked to actually update the display.
<b>Compatibility</b>	Implemented only if <a href="#">3.5 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">WinScreenLock</a>

## WinScrollRectangle

<b>Purpose</b>	Scroll a rectangle in the draw window.						
<b>Declared In</b>	Window.h						
<b>Prototype</b>	void WinScrollRectangle (const RectangleType *rP, WinDirectionType direction, Coord distance, RectangleType *vacatedP)						
<b>Parameters</b>	<table><tr><td>-&gt; rP</td><td>Pointer to the rectangle to scroll.</td></tr><tr><td>-&gt; direction</td><td>Direction to scroll (winUp, winDown, winLeft, or winRight).</td></tr><tr><td>-&gt; distance</td><td>Distance to scroll in pixels.</td></tr></table>	-> rP	Pointer to the rectangle to scroll.	-> direction	Direction to scroll (winUp, winDown, winLeft, or winRight).	-> distance	Distance to scroll in pixels.
-> rP	Pointer to the rectangle to scroll.						
-> direction	Direction to scroll (winUp, winDown, winLeft, or winRight).						
-> distance	Distance to scroll in pixels.						

## Windows

### Window Functions

---

`<- vacatedP` Pointer to the rectangle that needs to be redrawn because it has been vacated as a result of the scroll.

**Result** Returns nothing.

**Comments** The rectangle scrolls within its own bounds. Any portion of the rectangle that is scrolled outside its bounds is clipped.

## WinSetActiveWindow

**Purpose** Make a window the active window.

**Declared In** Window.h

**Prototype** void WinSetActiveWindow (WinHandle winHandle)

**Parameters** -> winHandle Handle of a window.

**Result** Returns nothing.

**Comments** The active window is not actually set in this routine; flags are set to indicate that a window is being exited and another window is being entered. The routine EvtGetEvent sends a [winExitEvent](#) and a [winEnterEvent](#) when it detects these flags. The active window is set by EvtGetEvent when it sends the winEnterEvent. The draw window is also set to the new active window when the active window is changed.

The window is enabled before it is made active.

All user input is directed to the active window.

**See Also** [WinGetActiveWindow](#), [EvtGetEvent](#)

## WinSetBackColor

**Purpose** Set the background color to use in subsequent draw operations.

**Declared In** Window.h

**Prototype** IndexedColorType WinSetBackColor  
(IndexedColorType backColor)

**Parameters** -> backColor Color to set; specify a value of type  
[IndexedColorType](#).

**Result** Returns the previous background color index.

**Comments** This function changes the current drawing state. If necessary, use [WinPushDrawState](#) to preserve the current drawing state before you set this function and use [WinPopDrawState](#) to restore it later. To set the foreground color to a predefined UI color default, use [UIColorGetTableEntryIndex](#) as an input to this function. For example:

```
curColor = WinSetBackColor  
(UIColorGetTableEntryIndex(UIFieldBackground)) ;
```

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [WinSetForeColor](#), [WinSetTextColor](#)

## Windows

### Window Functions

---

## WinSetBackColorRGB

**Purpose** Set the background color to use in subsequent draw operations.

**Declared In** Window.h

**Prototype**

```
void WinSetBackColorRGB
(const RGBColorType *newRgbP,
RGBColorType *prevRgbP)
```

**Parameters**

-> newRgbP	Color to set; specify a value of type <a href="#">RGBColorType</a> .
------------	---

<- prevRgbP	Previous color; specify a value of type <a href="#">RGBColorType</a> .
-------------	---

**Result** Returns nothing

**Comments** This function takes new and previous [RGBColorType](#) arguments. It is okay to set newRgbP or prevRgbP to NULL. If an application only wants to get the current color, the newRgbP argument is set to NULL. If the application does not care about the previous color, prevRgbP can be set to NULL.

This function sets the backColorRGB field of the [DrawStateType](#) structure to the value specified by newRgbP. It then sets the index field of backColorRGB to the 8 bit system palette entry that most closely matches the RGB components. Finally, it sets the backColor index field of [DrawStateType](#) to this index value.

This function changes the current drawing state. If necessary, use [WinPushDrawState](#) to preserve the current drawing state before you set this function and use [WinPopDrawState](#) to restore it later.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [WinSetForeColorRGB](#), [WinSetTextColorRGB](#)

## WinSetBounds

<b>Purpose</b>	Set the bounds of the window to display-relative coordinates.				
<b>Declared In</b>	Window.h				
<b>Prototype</b>	<code>void WinSetBounds (WinHandle winHandle, const RectangleType *rP)</code>				
<b>Parameters</b>	<table><tr><td>- &gt; winHandle</td><td>Handle for the window for which to set the bounds.</td></tr><tr><td>- &gt; rP</td><td>Pointer to a rectangle to use for bounds.</td></tr></table>	- > winHandle	Handle for the window for which to set the bounds.	- > rP	Pointer to a rectangle to use for bounds.
- > winHandle	Handle for the window for which to set the bounds.				
- > rP	Pointer to a rectangle to use for bounds.				
<b>Result</b>	Returns nothing.				
<b>Comments</b>	A visible window cannot have its bounds modified.				
<b>Compatibility</b>	Implemented only if <a href="#">2.0 New Feature Set</a> is present.				
<b>See Also</b>	<a href="#">WinGetBounds</a>				

## WinSetClip

<b>Purpose</b>	Set the clipping rectangle of the draw window.		
<b>Declared In</b>	Window.h		
<b>Prototype</b>	<code>void WinSetClip (const RectangleType *rP)</code>		
<b>Parameters</b>	<table><tr><td>- &gt; rP</td><td>Pointer to a structure holding the clipping bounds.</td></tr></table>	- > rP	Pointer to a structure holding the clipping bounds.
- > rP	Pointer to a structure holding the clipping bounds.		
<b>Result</b>	Returns nothing.		
<b>See Also</b>	<a href="#">WinClipRectangle</a> , <a href="#">WinSetClip</a> , <a href="#">WinGetClip</a>		

## Windows

### Window Functions

---

## WinSetDrawMode

**Purpose** Set the transfer mode to use in subsequent draw operations.

**Declared In** Window.h

**Prototype** WinDrawOperation WinSetDrawMode  
(WinDrawOperation newMode)

**Parameters** -> newMode Transfer mode to set; specify one of the [WinDrawOperation](#) values.

**Result** Returns the previous transfer mode.

**Comments** This function changes the current drawing state. If necessary, use [WinPushDrawState](#) to preserve the current drawing state before you set this function and use [WinPopDrawState](#) to restore it later.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.



## WinSetCoordinateSystem

---

**Purpose** Establish the coordinate system to be used for subsequent drawing operations.

**Declared In** Window.h

**Prototype** UInt16 WinSetCoordinateSystem (UInt16 coordSys)

**Parameters** -> coordSys The desired coordinate system. Supply one of the values defined in “[Window Coordinate System Constants](#)” on page 1163.

**Result** Returns the previous coordinate system value.

<b>Comments</b>	This function modifies the scale field in the draw state (a <a href="#">DrawStateType</a> structure). As when making other modifications to a window's draw state, applications should call <a href="#">WinPushDrawState</a> before modifying the coordinate system. To restore the coordinate system, your application can then call <a href="#">WinPopDrawState</a> .  To calculate the draw state scale field, the Window Manager divides the density of the bitmap associated with the draw window by coordSys. If coordSys is kCoordinatesNative, the Window Manager sets the scale field to 1.0, which enables 1-to-1 mapping of coordinates to pixels.  If you supply a value of kCoordinatesStandard for coordSys, subsequent drawing will use the standard coordinate system.
<b>Compatibility</b>	Implemented only if the <a href="#">High-Density Display Feature Set</a> is present.
<b>See Also</b>	<a href="#">WinGetCoordinateSystem</a>

## WinSetDrawWindow

<b>Purpose</b>	Set the draw window. (All drawing operations are relative to the draw window.)
<b>Declared In</b>	Window.h
<b>Prototype</b>	WinHandle WinSetDrawWindow (WinHandle winHandle)
<b>Parameters</b>	-> winHandle Handle of a window.
<b>Result</b>	Returns the previous draw window.
<b>Compatibility</b>	OS versions before 3.5 allowed you to use NULL as a parameter to this function to set the draw window to the display window (or screen window). In version 3.5 and higher, this practice is discouraged. If winHandle is NULL, the debug ROM sets the draw

## Windows

### Window Functions

---

window to badDrawWindowValue, and you are warned if you try to draw to it.

**See Also** [WinGetDrawWindow](#), [WinSetActiveWindow](#)

## WinSetForeColor

**Purpose** Set the foreground color to use in subsequent draw operations.

**Declared In** Window.h

**Prototype** IndexedColorType WinSetForeColor  
(IndexedColorType foreColor)

**Parameters** -> foreColor Color to set; specify a value of type [IndexedColorType](#).

**Result** Returns the previous foreground color index.

**Comments** This function changes the current drawing state. If necessary, use [WinPushDrawState](#) to preserve the current drawing state before you set this function and use [WinPopDrawState](#) to restore it later.

To set the foreground color to a predefined UI color default, use [UIColorGetTableEntryIndex](#) as an input to this function. For example:

```
curColor = WinSetForeColor  
          (UIColorGetTableEntryIndex  
           (UIObjectForeground));
```

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [WinSetBackColor](#), [WinSetTextColor](#)

## WinSetForeColorRGB

**Purpose** Set the foreground color to use in subsequent draw operations.

**Declared In** Window.h

**Prototype**

```
void WinSetForeColorRGB
(const RGBColorType *newRgbP,
RGBColorType *prevRgbP)
```

**Parameters**

- > newRgbP	Color to set; specify a value of type <a href="#">RGBColorType</a> .
< - prevRgbP	Previous color; specify a value of type <a href="#">RGBColorType</a> .

**Result** Returns nothing.

**Comments** This function takes new and previous [RGBColorType](#) arguments. It is okay to set newRgbP or prevRgbP to NULL. If an application only wants to get the current color, the newRgbP argument is set to NULL. If the application does not care about the previous color, prevRgbP can be set to NULL.

This function sets the foreColorRGB field of the [DrawStateType](#) structure to the value specified by newRgbP. It then sets the index field of foreColorRGB to the 8 bit system palette entry that most closely matches the RGB components. Finally, it sets the foreColor index field of [DrawStateType](#) to this index value.

This function changes the current drawing state. If necessary, use [WinPushDrawState](#) to preserve the current drawing state before you set this function and use [WinPopDrawState](#) to restore it later.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [WinSetBackColorRGB](#), [WinSetTextColorRGB](#)

## Windows

### Window Functions

---

## WinSetPattern

**Purpose** Set the current fill pattern.

**Declared In** Window.h

**Prototype** void WinSetPattern  
(const CustomPatternType \*patternP)

**Parameters** -> patternP Pattern to set (see [CustomPatternType](#)).

**Result** Returns nothing.

**Comments** The fill pattern is used by [WinFillLine](#) and [WinFillRectangle](#). This function changes the current drawing state. If necessary, use [WinPushDrawState](#) to preserve the current drawing state before you set this function and use [WinPopDrawState](#) to restore it later.

**See Also** [WinGetPattern](#)

## WinSetPatternType

**Purpose** Set the current pattern type.

**Declared In** Window.h

**Prototype** void WinSetPatternType (PatternType newPattern)

**Parameters** -> newPattern Pattern type to set for the draw window (see [PatternType](#)).

**Result** Returns nothing.

**Comments** This function sets the pattern field of the drawing state to newPattern and sets the patternData field to NULL. To set patternData to a custom pattern use [WinSetPattern](#).

The fill pattern is used by [WinFillLine](#) and [WinFillRectangle](#).

This function changes the current drawing state. If necessary, use [WinPushDrawState](#) to preserve the current drawing state before you set this function and use [WinPopDrawState](#) to restore it later.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [WinGetPatternType](#)

## WinSetTextColor

**Purpose** Set the color to use for drawing characters in subsequent draw operations.

**Declared In** Window.h

**Prototype** IndexedColorType WinSetTextColor  
(IndexedColorType textColor)

**Parameters** -> textColor Color to set; specify a value of type [IndexedColorType](#).

**Result** Returns the previous text color index.

**Comments** This function changes the current drawing state. If necessary, use [WinPushDrawState](#) to preserve the current drawing state before you set this function and use [WinPopDrawState](#) to restore it later.

To set the foreground color to a predefined UI color default, use [UIColorGetTableEntryIndex](#) as an input to this function. For example:

```
curColor = WinSetTextColor  
          (UIColorGetTableEntryIndex(UIFieldText)) ;
```

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

**See Also** [WinSetBackColor](#), [WinSetForeColor](#)

## Windows

### Window Functions

---

## WinSetTextColorRGB

**Purpose** Set the color to use for drawing characters in subsequent draw operations.

**Declared In** Window.h

**Prototype**

```
void WinSetTextColorRGB
(const RGBColorType *newRgbP,
RGBColorType *prevRgbP)
```

**Parameters**

-> newRgbP	Color to set; specify a value of type <a href="#">RGBColorType</a> .
<- prevRgbP	Previous color; specify a value of type <a href="#">RGBColorType</a> .

**Result** Returns nothing.

**Comments** This function takes new and previous [RGBColorType](#) arguments. It is okay to set newRgbP or prevRgbP to NULL. If an application only wants to get the current color, the newRgbP argument is set to NULL. If the application does not care about the previous color, prevRgbP can be set to NULL.

This function sets the textColorRGB field of the [DrawStateType](#) structure to the value specified by newRgbP. It then sets the index field of textColorRGB to the 8 bit system palette entry that most closely matches the RGB components. Finally, it sets the textColor index field of [DrawStateType](#) to this index value.

This function changes the current drawing state. If necessary, use [WinPushDrawState](#) to preserve the current drawing state before you set this function and use [WinPopDrawState](#) to restore it later.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [WinSetBackColorRGB](#), [WinSetForeColorRGB](#)

## WinSetUnderlineMode

**Purpose** Set the graphic state to enable or disable the underlining of characters.

**Declared In** Window.h

**Prototype** UnderlineModeType WinSetUnderlineMode  
(UnderlineModeType mode)

**Parameters** <-> mode      New underline mode type; see [UnderlineModeType](#).

**Result** Returns the previous underline mode type.

**Comments** This function changes the current drawing state. If necessary, use [WinPushDrawState](#) to preserve the current drawing state before you set this function and use [WinPopDrawState](#) to restore it later.

**See Also** [WinDrawChars](#)



**New**

## WinUnscaleCoord

**Purpose** Convert a single coordinate from the active coordinate system to the standard coordinate system.

**Declared In** Window.h

**Prototype** Coord WinUnscaleCoord (Coord coord,  
Boolean ceiling)

**Parameters** -> coord      A coordinate in the active coordinate system.  
-> ceiling      Pass true to round up, false to truncate the fractional part when scaling.

**Result** Returns the coordinate scaled to the standard coordinate system.

## Windows

### Window Functions

---

<b>Comments</b>	This function converts a coordinate by dividing it by the coordinate scaling factor, truncating or rounding the result to an integer value depending on the value of <code>ceiling</code> .  If the active coordinate system is <code>kCoordinatesStandard</code> , the returned coordinate is equal to the supplied coordinate.
<b>Compatibility</b>	Implemented only if the <a href="#">High-Density Display Feature Set</a> is present.
<b>See Also</b>	<a href="#">WinScaleCoord</a> , <a href="#">WinUnscalePoint</a> , <a href="#">WinUnscaleRectangle</a>



## WinUnscalePoint

---

<b>Purpose</b>	Convert a point from the active coordinate system to the standard coordinate system.					
<b>Declared In</b>	<code>Window.h</code>					
<b>Prototype</b>	<code>void WinUnscalePoint (PointType *pointP, Boolean ceiling)</code>					
<b>Parameters</b>	<table><tr><td><code>&lt;-&gt; pointP</code></td><td>Pointer to a <code>PointType</code> structure that, before the call, should contain a point's coordinates using the active coordinate system. After this function is called the <code>PointType</code> structure contains the coordinates of the point scaled to the standard coordinate system.</td></tr><tr><td><code>-&gt; ceiling</code></td><td>Pass <code>true</code> to round up, <code>false</code> to truncate the fractional part when scaling.</td></tr></table>		<code>&lt;-&gt; pointP</code>	Pointer to a <code>PointType</code> structure that, before the call, should contain a point's coordinates using the active coordinate system. After this function is called the <code>PointType</code> structure contains the coordinates of the point scaled to the standard coordinate system.	<code>-&gt; ceiling</code>	Pass <code>true</code> to round up, <code>false</code> to truncate the fractional part when scaling.
<code>&lt;-&gt; pointP</code>	Pointer to a <code>PointType</code> structure that, before the call, should contain a point's coordinates using the active coordinate system. After this function is called the <code>PointType</code> structure contains the coordinates of the point scaled to the standard coordinate system.					
<code>-&gt; ceiling</code>	Pass <code>true</code> to round up, <code>false</code> to truncate the fractional part when scaling.					
<b>Result</b>	Returns nothing. The coordinates of the point indicated by <code>pointP</code> are converted to the standard coordinate system.					
<b>Comments</b>	This function converts a point by dividing its x and y coordinates by the coordinate scaling factor, truncating or rounding the results to integer values depending on the value of <code>ceiling</code> . For instance,					

the input coordinates (11, 13) are transformed to (6, 7) if the input values represent native coordinates on a handheld with a double-density screen and ceiling is set to `true`. If `ceiling` is set to `false`, the same input coordinates are transformed to (5, 6).

If the active coordinate system is `kCoordinatesStandard`, `pointP` is not changed by this function.

**Compatibility** Implemented only if the [High-Density Display Feature Set](#) is present.

**See Also** [WinScalePoint](#), [WinUnscaleCoord](#), [WinUnscaleRectangle](#)



## WinUnscaleRectangle

**Purpose** Convert a rectangle from the active coordinate system to the standard coordinate system.

**Declared In** Window.h

**Prototype** void WinUnscaleRectangle (RectangleType \*rectP)

**Parameters** <-> rectP Pointer to a RectangleType structure that, before the call, should contain a rectangle's coordinates using the active coordinate system. After this function is called the RectangleType structure contains the coordinates of the rectangle scaled to the standard coordinate system.

**Result** Returns nothing. The coordinates of the rectangle indicated by `rectP` are converted to the standard coordinate system.

**Comments** This function scales the rectangle's `topLeft` and `extent` points by dividing their `x` and `y` coordinates by the coordinate scaling factor. All values are then truncated, but if either `topLeft.x` or `extent.x` had a fractional part, `extent.x` is incremented by 1

## Windows

### Window Functions

---

(and, similarly, if either `topLeft.y` or `extent.y` had a fractional part, `extent.y` is incremented by 1).

If the active coordinate system is `kCoordinatesStandard`, `rectP` is not changed by this function.

**Compatibility** Implemented only if the [High-Density Display Feature Set](#) is present.

**See Also** [WinScaleRectangle](#), [WinUnscaleCoord](#), [WinUnscalePoint](#)

## WinValidateHandle

**Purpose** Return `true` if the specified handle references a valid window object.

**Declared In** `Window.h`

**Prototype** `Boolean WinValidateHandle (WinHandle winHandle)`

**Parameters** `-> winHandle` The handle to be tested.

**Result** Returns `true` if the specified handle references a non-NULL pointer to a window in the active window list, `false` if the handle references a window whose values are out of sync with the current system state.

**Comments** For debugging purposes only. Do not include this function in commercial applications.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [FrmValidatePtr](#), [FrmRemoveObject](#)

## **WinWindowToDisplayPt**

**Purpose** Convert a window-relative coordinate to a display-relative coordinate.

**Declared In** Window.h

**Prototype** void WinWindowToDisplayPt (Coord \*extentX,  
Coord \*extentY)

**Parameters** <-> extentX Pointer to x coordinate to convert.  
<-> extentY Pointer to y coordinate to convert.

**Result** Returns nothing.

**Comments** The coordinate passed is assumed to be relative to the draw window.

**See Also** [WinDisplayToWindowPt](#)

## **Windows**

### *Window Functions*

---

# Miscellaneous System Functions

---

This chapter describes miscellaneous system functions. The functions in this chapter are declared in the header files `Crc.h`, `DLSERVER.h`, `IntlMgr.h`, and `Localize.h`.

## **Crc16CalcBlock**

**Purpose** Calculate the 16-bit CRC of a data block using the table lookup method.

**Declared In** `Crc.h`

**Prototype** `UInt16 Crc16CalcBlock (const void *bufP,  
                  UInt16 count, UInt16 crc)`

**Parameters** `bufP` Pointer to the data buffer.  
`count` Number of bytes in the buffer.  
`crc` Seed crc value.

**Result** A 16-bit CRC for the data buffer.

### DlkControl

<b>Purpose</b>	Perform an operation at the behest of the desktop software. Among other things, this function is used to return values to the conduit during the handling of <a href="#">sysAppLaunchCmdHandleSyncCallApp</a> .						
<b>Declared In</b>	<code>DLServer.h</code>						
<b>Prototype</b>	<code>Err DlkControl (DlkCtlEnum op, void *param1P, void *param2P)</code>						
<b>Parameters</b>	<table><tr><td>-&gt; op</td><td>Desktop Link control code. Use <code>d1kCtlSendCallAppReply</code> when sending a result back to the conduit while handling a <code>sysAppLaunchCmdHandleSyncCallApp</code> launch code.</td></tr><tr><td>&lt;-&gt; param1P</td><td>Pointer to the first parameter (operation-specific). For <code>d1kCtlSendCallAppReply</code>, this parameter should point to a <a href="#">DlkCallAppReplyParamType</a> structure.</td></tr><tr><td>&lt;-&gt; param2P</td><td>Pointer to the second parameter (operation-specific). For <code>d1kCtlSendCallAppReply</code>, this parameter should be set to NULL.</td></tr></table>	-> op	Desktop Link control code. Use <code>d1kCtlSendCallAppReply</code> when sending a result back to the conduit while handling a <code>sysAppLaunchCmdHandleSyncCallApp</code> launch code.	<-> param1P	Pointer to the first parameter (operation-specific). For <code>d1kCtlSendCallAppReply</code> , this parameter should point to a <a href="#">DlkCallAppReplyParamType</a> structure.	<-> param2P	Pointer to the second parameter (operation-specific). For <code>d1kCtlSendCallAppReply</code> , this parameter should be set to NULL.
-> op	Desktop Link control code. Use <code>d1kCtlSendCallAppReply</code> when sending a result back to the conduit while handling a <code>sysAppLaunchCmdHandleSyncCallApp</code> launch code.						
<-> param1P	Pointer to the first parameter (operation-specific). For <code>d1kCtlSendCallAppReply</code> , this parameter should point to a <a href="#">DlkCallAppReplyParamType</a> structure.						
<-> param2P	Pointer to the second parameter (operation-specific). For <code>d1kCtlSendCallAppReply</code> , this parameter should be set to NULL.						
<b>Result</b>	<code>errNone</code> if no error, or an error code if there was a problem during the call to <code>DlkControl</code> . In either case, place the value returned from <code>DlkControl</code> into the <code>replyErr</code> field of the <code>SysAppLaunchCmdHandleSyncCallAppType</code> structure when handling <code>sysAppLaunchCmdHandleSyncCallApp</code> .						
<b>Comments</b>	This function is needed to return data back to a conduit during the handling of <a href="#">sysAppLaunchCmdHandleSyncCallApp</a> . Set <code>param1P</code> to point to a <a href="#">DlkCallAppReplyParamType</a> structure, as described below. See the <a href="#">Example</a> on page 1249 for an illustration of how to handle <code>sysAppLaunchCmdHandleSyncCallApp</code> .						

### DlkCallAppReplyParamType

**Prototype**

```
typedef struct DlkCallAppReplyParamType {
    UInt16 pbSize;
    UInt32 dwresultCode;
    const void *resultP;
    UInt32 dwResultSize;
    void *dlRefP;
    UInt32 dwReserved1;
} DlkCallAppReplyParamType;
```

<b>Fields</b>	pbSize	Size of this parameter block. Set it to sizeof(DlkCallAppReplyParamType).
	dwresultCode	Result code to be returned to the remote caller.
	resultP	Pointer to result data.
	dwResultSize	Size of result data block (number of bytes).
	dlRefP	Desktop Link reference pointer from SysAppLaunchCmdHandleSyncCallAppType.
	dwReserved1	Reserved. Set to NULL.

**Example** The SysAppLaunchCmdHandleSyncCallAppType structure that accompanies the [sysAppLaunchCmdHandleSyncCallApp](#) launch code contains all of the information passed into SyncCallRemoteModule on the desktop as well as the necessary fields to pass the result pack to the desktop. At the end of your sysAppLaunchCmdHandleSyncCallApp launch code handler, you'll need to send a [DlkCallAppReplyParamType](#) reply structure back to the device using DlkControl.

---

```
#include <DLServer.h>
...
case sysAppLaunchCmdHandleSyncCallApp:
{
    SysAppLaunchCmdHandleSyncCallAppType *theCommandPtr;
    DlkCallAppReplyParamType theReplyParams;
    CharPtr theReplyBuffer = "SUCCESS";

    // Cast the cmdPBP to a SysAppLaunchCmdHandleSyncCallAppType
    // pointer so that we can work with it.
    theCommandPtr = (SysAppLaunchCmdHandleSyncCallAppType*) cmdPBP;
```

## Miscellaneous System Functions

---

```
// Do whatever work is necessary here. If you set the m_wActionCode
// field in your CCallModuleParams class on the desktop, then you
// can handle that code by looking at the action field of theCommandPtr
// (i.e.) if (theCommandPtr->action == 1)

// Create the reply to send back to the desktop

// First clear out all the fields. This is necessary so that the reserved
// fields are set to NULL.
MemSet( &theReplyParams, sizeof(DlkCallAppReplyParamType), 0 );

// Set the size of the reply. This is required.
theReplyParams.pbSize = sizeof(DlkCallAppReplyParamType);

// Set the result code. Normally this will be set to zero unless you want
// to send an error code back to the desktop.
theReplyParams.dwResultCode = 0;

// Fill in the reply buffer and buffer length
theReplyParams.resultP = theReplyBuffer;
theReplyParams.dwResultSize = StrLen(theReplyBuffer) + 1;

// Fill in the DL reference pointer. This is required.
theReplyParams.dlRefP = theCommandPtr->dlRefP;

// Set the handled field to true. This is required to let the desktop
// know that the sysAppLaunchCmdHandleSyncCallApp was handled. If you
// don't set this to true, the call to SyncCallRemoteModule will return
// SYNCERR_UNKNOWN_REQUEST.
theCommandPtr->handled = true;

// Finally, set the replyErr field by passing the reply parameters to
// DlkControl. This is required for the DLServer to properly handle the
// reply request.
theCommandPtr->replyErr = DlkControl (dlkCtlSendCallAppReply,
    &theReplyParams, NULL);

break;
}
```

---

[Table 55.1](#) and [Table 55.2](#) list some important mappings from the CCallModuleParams class on the desktop to the SysAppLaunchCmdHandleSyncCallAppType and DlkCallAppReplyParamType structures on the handheld.

**Table 55.1 CCallModuleParams to SysAppLaunchCmdHandleSyncCallAppType mapping**

CCallModuleParams	SysAppLaunchCmdHandleSyncCallAppType
m_wActionCode	action
m_dwParamSize	dwParamSize
m_pParam	paramP

**Table 55.2 CCallModuleParams to DLkCallAppReplyParamType mapping**

CCallModuleParams	DLkCallAppReplyParamType
m_dwResultBufSize	dwResultSize
m_pResultBuf	resultP
m_dwresultCode	dwresultCode

## DLkGetSyncInfo

**Purpose** Get the sync info managed by Desktop Link. This function is often used to obtain the user name on the handheld.

**Declared In** DLServer.h

**Prototype**

```
Err DlkGetSyncInfo (UInt32 *succSyncDateP,
                    UInt32 *lastSyncDateP,
                    DLkSyncStateType *syncStateP, Char *nameBufP,
                    Char *logBufP, Int32 *logLenP)
```

**Parameters**

<- succSyncDateP	Pointer to the location where the date of the last successful sync is stored. Supply NULL for this parameter if this date isn't needed.
------------------	---

## Miscellaneous System Functions

---

<- lastSyncDateP	Pointer to the location where the date of the last sync, successful or otherwise, is stored. Supply NULL for this parameter if this date isn't needed.
<- syncStateP	Pointer to a DlkSyncStateType enum into which the state of the last sync is stored. Supply NULL for this parameter if the state information isn't needed. See the Comments, below, for a description of this enum.
<- nameBufP	Pointer to a string buffer into which the null-terminated handheld user name is stored. This string buffer must have been preallocated to be at least dlkUserNameBufSize bytes in length. Supply NULL for this parameter if the user name isn't needed.
<- logBufP	Pointer to a string buffer into which the sync log text, null-terminated, is stored. Supply NULL for this parameter if the log text isn't needed. If you supply a valid pointer for this parameter, you must specify the preallocated buffer length using the logLenP parameter; the returned log text will be truncated, if necessary, to fit within the buffer.
<-> logLenP	Pointer to the log buffer size. If logBufP is not NULL, on entry you must set this value to the size of the logBufP buffer. When this function returns, this value indicates the actual length of the log text, not counting the null terminator.

**Result** Returns errNone if no error, or dlkErrMemory if the Desktop Link preferences resource couldn't be locked.

**Comments** The state information returned through syncStateP has one of the values defined by the DlkSyncStateType enum:

```
typedef enum DlkSyncStateType {  
    dlkSyncStateNeverSynced = 0,  
    dlkSyncStateInProgress,
```

```
dlkSyncStateLostConnection,  
dlkSyncStateLocalCan,  
dlkSyncStateRemoteCan,  
dlkSyncStateLowMemoryOnTD,  
dlkSyncStateAborted,  
dlkSyncStateCompleted,  
dlkSyncStateIncompatibleProducts,  
dlkSyncStateNPOD  
} DlkSyncStateType;
```

Value	Description
dlkSyncStateNeverSynced	The handheld has never been synced.
dlkSyncStateInProgress	A sync is currently in progress.
dlkSyncStateLostConnection	The connection was lost during sync.
dlkSyncStateLocalCan	Sync was cancelled by the user on the handheld.
dlkSyncStateRemoteCan	Sync was cancelled by the user from the desktop.
dlkSyncStateLowMemoryOnTD	Sync ended due to a low memory condition on the handheld.
dlkSyncStateAborted	Sync was aborted for some other reason.
dlkSyncStateCompleted	Sync completed normally.

## Miscellaneous System Functions

---

Value	Description
dlkSyncStateIncompatibleProducts	Sync ended because the desktop HotSync product is incompatible with this version of the handheld HotSync.
dlkSyncStateNPOD	The sync could not take place because the handheld has a 4.0-style password but the desktop hasn't yet been updated to a compatible version.

**Example** This function is most often used to obtain the handheld user name. The following code excerpt shows how to do this (for clarity, error-checking has been omitted):

```
MemHandle nameH;
char *nameP;

// Allocate a buffer for the user name
nameH = MemHandleNew(dlkUserNameBufSize);
nameP = MemHandleLock(nameH);

// Obtain the user's name
DlkGetSyncInfo(NULL, NULL, NULL, nameP, NULL, NULL);

// ... Do something with the user name here ...

// Now that we're done with the user name, free the buffer
MemPtrUnlock(nameP);
```

---

**Compatibility** The `dlkSyncStateIncompatibleProducts` enum value was added in Palm OS 3.0. The `dlkSyncStateNPOD` enum value was added in Palm OS 4.0.

### IntlGetRoutineAddress

<b>Purpose</b>	Return the address of an international manager or text manager function.
<b>Declared In</b>	<code>IntlMgr.h</code>
<b>Prototype</b>	<code>void *IntlGetRoutineAddress (IntlSelector inSelector)</code>
<b>Parameters</b>	<code>-&gt; inSelector</code> One of the routine selectors defined in <code>IntlMgr.h</code> .
<b>Result</b>	Returns the address of the corresponding function. Returns NULL if an invalid routine selector is passed.
<b>Comments</b>	<p>Use this function for performance reasons. It returns the address of an international manager or text manager function. You can then use this address to call the function without having to go through the international manager's trap dispatch table. This function is mostly useful for optimizing the performance of text manager routines that are called in a tight loop.</p> <p>You might also use this function to check for the presence of newer international manager and text manager functions. If the result is NULL, the function is not implemented on this device.</p>
<b>Compatibility</b>	Implemented only if <a href="#">International Feature Set</a> is present.
<b>See Also</b>	<a href="#">IntlSetRoutineAddress</a> , <a href="#">SysGetTrapAddress</a>

### IntlSetRoutineAddress

<b>Purpose</b>	Set the address of the function corresponding to an international manager or text manager function.				
<b>Declared In</b>	IntlMgr.h				
<b>Prototype</b>	<pre>Err IntlSetRoutineAddress (IntlSelector iSelector, void *iProcPtr)</pre>				
<b>Parameters</b>	<table><tr><td>-&gt; iSelector</td><td>One of the routine selectors defined in IntlMgr.h.</td></tr><tr><td>-&gt; iProcPtr</td><td>Pointer to a function that the routine identified by iSelector should point to.</td></tr></table>	-> iSelector	One of the routine selectors defined in IntlMgr.h.	-> iProcPtr	Pointer to a function that the routine identified by iSelector should point to.
-> iSelector	One of the routine selectors defined in IntlMgr.h.				
-> iProcPtr	Pointer to a function that the routine identified by iSelector should point to.				
<b>Result</b>	Returns errNone if no error, or intlErrInvalidSelector if iSelector does not refer to a valid international manager or text manager routine.				
<b>Comments</b>	This function is useful for patching an international or text manager function. Normally only a locale module would need to patch one of these functions.				
<hr/> <b>WARNING!</b> If your application patches an international manager function using this function, you <b>must</b> remove the patch before your application exits. Do <b>not</b> use this mechanism to permanently patch international manager functions as it may cause unpredictable results for the system and other applications.					
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present. If <a href="#">5.0 New Feature Set</a> is present this function is unimplemented.				
<b>See Also</b>	<a href="#">IntlGetRoutineAddress</a> , <a href="#">SysSetTrapAddress</a>				

## LocGetNumberSeparators

**Purpose** Get localized number separators.

**Declared In** Localize.h

**Prototype** void LocGetNumberSeparators  
(NumberFormatType numberFormat,  
Char \*thousandSeparator, Char \*decimalSeparator)

**Parameters** -> numberFormat The format to use (see [NumberFormatType](#)).  
<- thousandSeparator  
The character used for the thousands separator.  
<- decimalSeparator  
The character used for the decimal separator.

**Result** Returns nothing.

**Comments** The format to use is stored in the system preferences. You can obtain it by passing prefNumberFormat to [PrefGetPreference](#).

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

**See Also** [StrLocalizeNumber](#), [StrDelocalizeNumber](#), “[Localized Applications](#)” in the *Palm OS Programmer’s Companion*, vol. I



### New PceNativeCall

**Purpose** Call a native ARM or Windows NT function from code running in the PACE (68k) environment.

**Declared In** PceNativeCall.h

**Prototype**

```
UInt32 PceNativeCall(  
    NativeFuncType *nativeFuncP, void *userDataP)
```

**Parameters**

-> nativeFuncP	On a handheld with an ARM processor, this is a pointer to the ARM function to be executed. On Palm OS Simulator, this is a pointer to the name of a DLL and the name of the entry point within that DLL that is to be executed, separated by a null character and terminated with a null character. See the Comments, below, for more details on the format of this argument.
----------------	---

<- > userDataP	Pointer to an application-specific block of data that is passed to the ARM function. This block has no specific alignment requirements; it needn't be aligned on a 16- or 32-bit boundary. Note that your ARM function may impose specific alignment requirements, however.
----------------	---

**Result** The return value of the specified ARM function is returned by PceNativeCall. This value is placed in both the A0 and D0 registers in the emulated 68k CPU, allowing PceNativeCall to support both pointer and immediate return value conventions.

**Comments** Applications that employ PceNativeCall won't work on handhelds running a version of Palm OS prior to Palm OS 5. Before calling PceNativeCall, your application must verify the underlying processor type, since the calling convention is different on Palm OS Simulator. See "[Calling an ARM Function](#)" on page 340 of *Palm OS Programmer's Companion*, vol. I for more information and an example.

PceNativeCall byte-swaps the parameter pointer and return value as appropriate for the Dragonball-to-ARM transition. This allows you to dereference userDataP directly from your ARM code. Because the operating system has no knowledge of the structure of the parameter block, however, it performs no byte-swapping within this block. Your ARM code must do this as necessary for your application (see “[Accessing 68K Data From an ARM Function](#)” on page 342 of the *Palm OS Programmer’s Companion*, vol. I for more information).

On Palm OS Simulator, rather than passing a pointer to a block of ARM code in nativeFuncP, you instead pass a pointer to the name of a DLL and the name of the function within that DLL that is to be executed. These two names must be separated by a null character, and the entire sequence must be terminated by a null character. For example, to load the DLL found at

C:\TEST\_DLL\Debug\Simple.dll and call the function TestNativeCall within that DLL, you might pass a pointer to the following character string literal:

```
"C:\\TEST_DLL\\\\Debug\\\\Simple.dll\\0TestNativeCall"
```

Note that if you don’t supply an absolute path, Simulator looks for the DLL in (or relative to) the directory from which PalmSim.exe is running. Thus, if the DLL is located in the same directory as PalmSim.exe, you can call the above function with:

```
"Simple.dll\\0TestNativeCall"
```

On release ROMs, PceNativeCall fails silently if nativeFuncP is NULL. On debug ROMs, it generates an error. All other pointers are treated as valid code and followed. If nativeFuncP is invalid, the processor will try to execute the code anyway and will eventually generate an error.

---

**NOTE:** The call to your native function is guaranteed to be made from ARM mode.

---

For more information on how the ARM code should be structured and how to call back and forth between the PACE and ARM environments, see “[ARM-Native Functions](#)” on page 339 of the *Palm OS Programmer’s Companion*, vol. I.

## Miscellaneous System Functions

---

**Compatibility** Implemented only if [5.0 New Feature Set](#) is present.

# **Part III: Communications**



# Connection Manager

---

The Connection Manager allows other applications to access, add, and delete connection profiles contained in the Connection panel.

This chapter provides reference material for the Connection Manager API declared in the header file `ConnectionMgr.h`:

- [Connection Manager Constants](#)
- [Connection Manager Functions](#)

For more information on the Connection Manager, see the chapter “[Serial Communication](#)” on page 89 in the *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## Connection Manager Data Types

### CncProfileID

The `CncProfileID` type uniquely identifies a connection profile within the Connection Manager profile database. You pass this ID as a parameter to several of the Connection Manager functions. You can obtain a connection’s profile ID using

[CncProfileGetIDFromName](#) or [CncProfileGetIDFromIndex](#).

```
typedef UInt32 CncProfileID
```

**Compatibility**    Defined only if [Connection Manager Feature Set](#) is present.

## Connection Manager Constants

### Profile Parameter Constants

The Connection Manager defines the following constants to represent individual parameters in the preinstalled connection profiles. Not all parameters work for all types of profiles.

## Connection Manager

### *Connection Manager Constants*

---

Parameter Name	Parameter Type	Description
kCncParamBaud	UInt32	The baud rate to use for the connection.
kCncParamBluetoothDevice Addr	buffer	The 48-bit address (BD_ADDR) of the device that the handheld is connected to through the Bluetooth port. This parameter is only valid if kCncParamPort specifies the Bluetooth port.
kCncParamBluetoothDevice Name	string	The name of the device that the handheld is connected to through the Bluetooth port. The device name uses the UTF-8 character encoding. This parameter is only valid if kCncParamPort specifies the Bluetooth port.
kCncParamCountryIndex	UInt16	The index into the list of strings returned by kCncParamIntlModemCountryStringList and kCncParamIntlModemResetStringList that provides the name of the country and the reset commands for this profile.
kCncParamDeviceKind	UInt16	The type of connection being made (general serial connection, connection to a modem, connection to a phone, and so on). This value is one of the <a href="#">Device Kind Constants</a> .
kCncParamDialingMode	UInt8	For modem profiles, the dialing mode. 1 for Pulse dialing, 0 for TouchTone.

kCncParamCountryIndex	UInt16	The index into the list of strings returned by kCncParamIntlModemCountryStringList and kCncParamIntlModemResetStringList that provides the name of the country and the reset commands for this profile.
kCncParamDeviceKind	UInt16	The type of connection being made (general serial connection, connection to a modem, connection to a phone, and so on). This value is one of the <a href="#">Device Kind Constants</a> .
kCncParamDialingMode	UInt8	For modem profiles, the dialing mode. 1 for Pulse dialing, 0 for TouchTone.

## Connection Manager

### *Connection Manager Constants*

---

Parameter Name	Parameter Type	Description
kCncParamIntlModemCountryStringList	buffer	<p>For modem profiles, a data buffer containing a list of possible countries. This list contains the countries in which the modem can operate and is shown in the Connection panel Details form for this profile. The selected country controls the dialing prefix and the modem reset string. The kCncParamCountryIndex parameter specifies the country selected from this list.</p> <p>The data buffer contains a packed block of null-terminated strings, each containing a country name. The first 16 bits of the data buffer (a UInt16 value) tells how many strings are contained in the block. You can use <a href="#">SysFormPointerArrayToString</a>s to convert the data buffer into an array of strings.</p> <p>Because this parameter value has a variable size, you must first call <a href="#">CncProfileSettingGet</a> with a NULL data pointer to obtain the correct size.</p>
kCncParamIntlModemResetStringList	buffer	For modem profiles, a data buffer containing possible reset strings. The kCncParamCountryIndex parameter specifies which reset string from this list is to be used. The actual string used is itself stored in the kCncParamResetString parameter.

Parameter Name	Parameter Type	Description
		The data buffer contains a packed block of null-terminated strings, each containing a reset string. The first 16 bits of the data buffer (a UInt16 value) tells how many strings are contained in the block. You can use <a href="#"><u>SysFormPointerArrayToString</u></a> to convert the data buffer into an array of strings.
		Because this parameter value has a variable size, you must first call <a href="#"><u>CncProfileSettingGet</u></a> with a NULL data pointer to obtain the correct size.
kCncParamInvisible	system flag	If true, the profile is hidden from the user. If false, the profile is visible. This parameter is not currently used.
kCncParamLocked	system flag	If true, the profile is locked so that the user cannot edit it. If false, the profile can be edited. This parameter can be set by profiles, such as phone profiles, created by third party utilities.
kCncParamName	string	The name of the profile.
kCncParamNoDetails	system flag	If true, the profile details should not be displayed. If false, they can be displayed. The profile details are the parameters that appear in the Details form of the Connection panel.

## Connection Manager

### *Connection Manager Constants*

---

Parameter Name	Parameter Type	Description
kCncParamNonEditable	system flag	If true, the Connection panel's Edit form should be suppressed for this profile. If false, the Edit form can be displayed.  This parameter differs from kCncParamLocked and kCncParamReadOnly in that it causes an alert to be displayed if the user taps the Edit button. The other parameters allow the Edit form to be displayed but do not allow changes to be made. Also, a non-editable profile can be deleted, but a read-only or locked profile cannot.
kCncParamPort	UInt32	The logical, physical, or virtual port identifier. See " <a href="#">Port Constants</a> " on <a href="#">page 1557</a> in the " <a href="#">Serial Manager</a> " chapter for more information.
kCncParam_PSDCreator	UInt32	For phone profiles, the creator ID of the phone driver.
kCncParam_PSDName	string	For phone profiles, the name of the phone driver.
kCncParam_PSDParameterBuffer	buffer	For phone profiles, a data buffer containing any necessary data that the phone driver needs to store. This parameter typically holds data that is set using the Details form of the Connection panel.
kCncParam_PSDType	UInt32	For phone profiles, the database type for the phone driver.

Parameter Name	Parameter Type	Description
kCncParamReadOnly	system flag	If <code>true</code> , the profile is read-only and cannot be edited. If <code>false</code> , the profile can be edited. This parameter is only intended to be used by profiles pre-installed in the Palm OS.
kCncParamReceiveTimeOut	UInt32	For phone profiles, the number of milliseconds to wait for a response from the phone. This time-out value is used by telephony functions that don't need to access the network (for example, the function <a href="#">TelNwkGetSelectedNetwork</a> ).
kCncParamResetString	string	For modem and phone profiles, the reset string. For modem profiles, this is one of the strings in <code>kCncParamIntlModemResetStringList</code> .
kCncParamSerialPortFlags	UInt32	For phone profiles, bit flags that correspond to various serial port hardware settings. See “ <a href="#">Serial Settings Constants</a> ” on page 1560 for more information.
kCncParamSystemFlags	UInt32	A bit flag representing all system flags. Currently, only bits 0 through 4 have a meaning. These correspond to the read-only bit, the invisible bit, the noneditable bit, the no details bit, and the locked bit, respectively.
kCncParamTimeOut	UInt32	The amount of time in milliseconds to wait for a response when CTS is unasserted and hardware flow control is on.

## Connection Manager

### *Connection Manager Constants*

---

Parameter Name	Parameter Type	Description
kCncParamTTCreator	UInt32	For phone profiles, the creator ID of the telephony task used by the phone driver.
kCncParamTTType	UInt32	For phone profiles, the database type for the telephony task used by the phone driver.
kCncParamVersion	UInt8	The version of the Connection Manager API under which this profile was created. The current version number is kCncProfileVersion.
kCncParamVolume	UInt16	For modem profiles, the modem volume.

**Compatibility** Defined only if [Connection Manager Feature Set](#) is present.

## Profile Parameter Size Constants

The size constants specify the size of each of the predefined parameters described in “[Profile Parameter Constants](#).” The following table lists the parameter name, the size of its value, and the size constant that gives this size. These size constants are suitable for passing to [CncProfileSettingGet](#) and [CncProfileSettingSet](#).

Profile Parameter Name	Size	Size Constant
kCncParamBaud	32	kCncParamBaudSize
kCncParamBluetoothDeviceAddr	8	kCncParamBluetoothDeviceAddrSize
kCncParamBluetoothDeviceName	249	kCncParamBluetoothDeviceNameMaxSize
kCncParamCountryIndex	16	kCncParamCountryIndexSize

## Connection Manager

*Connection Manager Constants*

---

<b>Profile Parameter Name</b>	<b>Size</b>	<b>Size Constant</b>
kCncParamDeviceKind	16	kCncParamDeviceKindSize
kCncParamDialingMode	8	kCncParamDialingModeSize
kCncParamFlowControl	16	kCncParamFlowControlSize
kCncParamInitString	81	kCncParamInitStringMaxSize
	81	kCncProfileUsualInitStringSize
kCncParamInvisible	8	kCncParamInvisibleSize
kCncParamLocked	8	kCncParamLockedSize
kCncParamName	22	kCncParamNameMaxSize
	22	kCncProfileNameSize
kCncParamNoDetails	8	kCncParamNoDetailsSize
kCncParamNonEditable	8	kCncParamNonEditableSize
kCncParamPort	32	kCncParamPortSize
kCncParam_PSDCreator	32	kCncParam_PSDCreatorSize
kCncParam_PSDName	32	kCncParam_PSDNameSize
kCncParam_PSDType	32	kCncParam_PSDTypeSize
kCncParamReadOnly	8	kCncParamReadOnlySize
kCncParamReceiveTimeOut	32	kCncParamReceiveTimeOutSize
kCncParamResetString	81	kCncParamResetStringMaxSize
	8	kCncProfileClassicResetStringSize
	81	kCncProfileUsualResetStringSize
kCncParamSystemFlags	32	kCncParamSystemFlagsSize
kCncParamTimeOut	32	kCncParamTimeOutSize

## Connection Manager

### *Connection Manager Constants*

---

Profile Parameter Name	Size	Size Constant
kCncParamTTCreator	32	kCncParamTTCreatorSize
kCncParamTTType	32	kCncParamTTTypeSize
kCncParamVersion	8	kCncParamVersionSize
kCncParamVolume	16	kCncParamVolumeSize

**Compatibility** Defined only if [Connection Manager Feature Set](#) is present.

## Device Kind Constants

The device kind constants specify the type of connection being made. You specify the type of connection by defining a value for the kCncParamDeviceKind parameter using [CncProfileSettingSet](#).

Constant	Value	Description
kCncDeviceKindSerial	0	The connection is through the serial port.
kCncDeviceKindModem	1	The connection is to a modem.
kCncDeviceKindPhone	2	The connection is to a phone.
kCncDeviceKindLocalNetwork	3	The connection is to a LAN.

**Compatibility** Defined only if [Connection Manager Feature Set](#) is present.

## Profile Parameter Types

The parameter type constants specify the type of value stored for a parameter in a connection profile. If you define a new parameter using [CncDefineParamID](#), you must use one of these constants to specify the type of data the parameter stores. The macro [CncGetParamType](#) can return this information for any parameter in the profile.

Constant	Value	Description
kCncParamSystemFlag	0x00	A system flag parameter. The Connection Manager can store up to 32 system flags. Flags are stored in a single bit and returned as a UInt8 value. The entire system flags word can be returned if you pass kCncParamSystemFlags to <a href="#"><u>CncProfileSettingGet</u></a> .
kCncParamUInt8	0x01	A UInt8 parameter.
kCncParamUInt16	0x02	A UInt16 parameter.
kCncParamUInt32	0x03	A UInt32 parameter.
kCncParamString	0x04	A string parameter.
kCncParamBuffer	0x05	A generic block of data.

**Compatibility** Defined only if [Connection Manager Feature Set](#) is present.

## Connection Manager Functions

### CncAddProfile

**Purpose** Adds a profile to the Connection Manager.

**Declared In** ConnectionMgr.h

**Prototype** Err CncAddProfile (Char \*name, UInt32 port,  
 UInt32 baud, UInt16 volume, UInt16 handShake,  
 const Char \*initString, const Char \*resetString,  
 Boolean isModem, Boolean isPulse)

**Parameters** <-> name      Pointer to the profile name to be added. If the name is already taken in the Connection panel then a duplication number is appended to it. The name added is returned here.

## Connection Manager

### Connection Manager Functions

---

-> port	The port identification used by the profile. See “ <a href="#">Specifying the Port</a> ” on page 100 of the <i>Palm OS Programmer’s Companion</i> , vol. II, <i>Communications</i> for more information.
-> baud	The baud rate used by the profile.
-> volume	The volume setting for the device (for Modem only).
-> handShake	Flow control setting (hardware handshaking). 0 specifies automatic (on at speeds > 2400 baud), 1 specifies always on, and 2 specifies always off.
-> initString	Pointer to the initialization string used by a modem (for Modem only).
-> resetString	Pointer to the reset string used by a modem (for Modem only).
-> isModem	true if Modem, false if Direct.
-> isPulse	true if Pulse dial, false if TouchTone.
<b>Result</b>	
errNone	No error.
cncErrAddProfileFailed	The add operation failed.
cncErrProfileListFull	The add operation failed because the profile list is full.
cncErrConDBNotFound	The connection database is missing.

**Comments** All profiles within the Connection Manager must have a unique name. The Connection Manager tries to append a duplication number to the end of the name if you specify a name that is already taken. There is a maximum limit to the number of profiles that can be maintained by the Connection Manager. If the limit is passed, an error is returned and that profile will not be added.

Profiles that do not need certain fields may pass NULL in the place of a value.

**Compatibility** Implemented only if [New Serial Manager Feature Set](#) is present.

If [Connection Manager Feature Set](#) is present, use [CncProfileCreate](#) instead of using this function.

`CncAddProfile` is still supported for backward compatibility. In Palm OS 4.0 and higher, the maximum number of profiles that can be defined has greatly increased.

**Example**

```
AddMyProfile()
{
    Char *myConNameP;
    Err err;

    myConNameP = MemPtrNew(cncProfileNameSize);

    StrCopy(myConNameP, "Foobar");

    err = CncAddProfile(myConNameP, 'u328', 57600, 0, 0,
                        "AT&FX4", 0, true, false);

    MemPtrFree(myConNameP);
}
```

## CncDefineParamID

**Purpose** Macro that creates and returns a parameter ID.

**Declared In** ConnectionMgr.h

**Prototype** CncDefineParamID (parameterRange, parameterType, parameterID)

**Parameters** -> parameterRange

Use `kCncParamThirdPartiesRange` to specify that this parameter is not defined by the OS.

## Connection Manager

### Connection Manager Functions

---

-> parameterType

The type of value stored for the parameter. See “[Profile Parameter Types](#)” for a list of possible values.

-> parameterID

A unique value between 0 to 1023. The value must be unique within the profile for which you are defining the parameter.

If you are using a parameterType of kCncParamSystemFlag, specify a value from 0 to 31 to identify which of the system flag bit is to be set.

**Result** Returns the parameter ID as a UInt16 value.

**Comments**

You use this macro only if you are defining your own connection profile and have a parameter that you need to define within that profile. The parameter ID immediately precedes its parameter value in the Connection Manager profile database. Because of how the database is formatted, the parameter ID must tell the Connection Manager how to interpret the next series of bytes. For this reason, the high order bits of the parameter ID include information about the type of value and whether the value is defined by the system or a third party.

**Compatibility**

Parameter IDs of this format are only used if [4.0 New Feature Set](#) is present.

**See Also**

[CncProfileSettingSet](#), [CncProfileSettingGet](#)

## CncDeleteProfile

<b>Purpose</b>	Removes a profile from the Connection Manager.
<b>Declared In</b>	ConnectionMgr.h
<b>Prototype</b>	Err CncDeleteProfile (const Char *name)
<b>Parameters</b>	-> name                    Pointer to the name of the profile to be deleted.
<b>Result</b>	errNone                    No error. cncErrProfileReadOnly                    The profile could not be deleted because it is read only. cncErrProfileNotFound                    The profile could not be found. cncErrConDBNotFound                    The connection database is missing.
<b>Comments</b>	The profiles that come preinstalled on the unit are read only and cannot be deleted.
<b>Compatibility</b>	Implemented only if <a href="#">New Serial Manager Feature Set</a> is present. If <a href="#">Connection Manager Feature Set</a> is present, use <a href="#">CncProfileDelete</a> instead of using this function.

## CncGetParamType

<b>Purpose</b>	Macro that returns the parameter type portion of the parameter ID.
<b>Declared In</b>	ConnectionMgr.h
<b>Prototype</b>	CncGetParamType (parameterID)
<b>Parameters</b>	-> parameterID A UInt16 that contains the parameter ID.
<b>Result</b>	Returns a UInt16 value where bits 11 through 14 contain one of the values in “ <a href="#">Profile Parameter Types</a> ” and the other bits are clear.
<b>Compatibility</b>	Parameter IDs of this format are only used if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">CncProfileSettingSet</a> , <a href="#">CncProfileSettingGet</a>

## CncGetProfileInfo

<b>Purpose</b>	Returns the settings for a profile.
<b>Declared In</b>	ConnectionMgr.h
<b>Prototype</b>	Err CncGetProfileInfo (Char *name, UInt32 *port, UInt32 *baud, UInt16 *volume, UInt16 *handShake, Char *initString, Char *resetString, Boolean *isModem, Boolean * isPulse)
<b>Parameters</b>	-> name Pointer to the name of the profile to be returned. Passing in NULL causes this function to return the settings for the profile currently selected in the Connection panel.  <- port Pointer to the port identifier that the profile uses.  <- baud Pointer to the baud rate that has been set for this profile.

<- volume	Pointer to the volume of the device (applies only to modems).
<- handShake	Pointer to the flow control setting (hardware handshaking). 0 indicates automatic (on at speeds > 2400 baud), 1 indicates always on, and 2 indicates always off.
<- initString	Pointer to the initialization string for the device (applies only to modems).
<- resetString	Pointer to the reset string for the device (applies only to modems).
<- isModem	Pointer to a Boolean value: <code>true</code> for Modem, <code>false</code> for Direct.
<- isPulse	Pointer to a Boolean value: <code>true</code> for Pulse dial, <code>false</code> for TouchTone.
<b>Result</b>	<p><code>errNone</code> No error.</p> <p><code>cncErrGetProfileFailed</code> The get profile operation failed. The profile may or may not be there.</p> <p><code>cncErrProfileNotFound</code> The profile could not be found</p> <p><code>cncErrConDBNotFound</code> The connection database is missing.</p>
<b>Comments</b>	One or more of the parameters may be set to NULL if that information is not desired.
<b>Compatibility</b>	Implemented only if <a href="#">New Serial Manager Feature Set</a> is present.  If <a href="#">Connection Manager Feature Set</a> is present, use <a href="#">CncProfileSettingGet</a> with one of the “ <a href="#">Profile Parameter Constants</a> ” instead of using this function.
<b>Example</b>	<pre>{     UInt32 portID, baud;     UInt16 openPort;     // get port id     err = CncGetProfileInfo("Direct Serial", &amp;portID, &amp;baud,</pre>

## Connection Manager

### *Connection Manager Functions*

---

```
    0, 0, 0, 0, 0, 0, 0);  
    if(!err)  
    { // open the port  
        SrmOpen(portID, baud, &openPort);  
    }  
}
```

## CncGetProfileList

**Purpose** Returns a list of available profiles that are available through the Connection Manager.

**Declared In** ConnectionMgr.h

**Prototype** Err CncGetProfileList (Char \*\*\*nameListPPP,  
                  UInt16 \*countP)

**Parameters** <- nameListPPP    Pointer to a pointer to a list of profile names.  
                  <- countP              Pointer to the number of profile names.



## Connection Manager

### Connection Manager Functions

---

<b>Result</b>	errNone	No error.
	cncErrGetProfileListFailed	The profile list could not be found.
	cncErrConDBNotFound	The connection database is missing.

**Comments** Allocation of the list is handled by the Connection Manager; deallocation is the responsibility of the calling application. Appended to the end of the list will be “-Current-”, which represents the profile currently selected in the Connection panel.

**Compatibility** Implemented only if [New Serial Manager Feature Set](#) is present.

**Example**

```
//Declared globally
Char ** globalProfileList;
ListType *listP;
UInt16 globalProfileCount;

void SetConnectionList()
{
    //Get the list from the Connection Manager
    err = CncGetProfileList(&globalProfileList,
                           &globalProfileCount);
    //Set the UI list
    LstSetListChoices(listP, globalProfileList,
                      globalProfileCount);
}

void StopApplication()
{
    UInt16 i;

    //Deallocate the connection list
    For(i = 0; i < globalProfileCount; i++)
        MemPtrFree(globalProfileList[ i ]);
    MemPtrFree(globalProfileList);
}
```

## CncGetSystemFlagBitnum

<b>Purpose</b>	Macro that returns the number uniquely identifying a system flag parameter.
<b>Declared In</b>	ConnectionMgr.h
<b>Prototype</b>	<code>CncGetSystemFlagBitnum (parameterID)</code>
<b>Parameters</b>	-> parameterID    The UInt16 containing the parameter ID.
<b>Result</b>	Returns the ID of the system flag parameter, which is a value from 0 to 31.
<b>Compatibility</b>	Parameter IDs of this format are only used if <a href="#">Connection Manager Feature Set</a> is present.
<b>See Also</b>	<a href="#">CncProfileSettingSet</a> , <a href="#">CncProfileSettingGet</a>

## CncGetTrueParamID

<b>Purpose</b>	Macro that returns the portion of the parameter ID that uniquely identifies the parameter.
<b>Declared In</b>	ConnectionMgr.h
<b>Prototype</b>	<code>CncGetTrueParamID (parameterID)</code>
<b>Parameters</b>	-> parameterID    A UInt16 containing the parameter ID.
<b>Result</b>	Returns a UInt16 containing just the parameter ID. The high-order bits, which specify the parameter type and address space, are clear.
<b>Compatibility</b>	Parameter IDs of this format are only used if <a href="#">Connection Manager Feature Set</a> is present.
<b>See Also</b>	<a href="#">CncProfileSettingSet</a> , <a href="#">CncProfileSettingGet</a>

## CncIsFixedLengthParamType

<b>Purpose</b>	Macro that specifies whether the parameter value is fixed length or variable length.
<b>Declared In</b>	ConnectionMgr.h
<b>Prototype</b>	<code>CncIsFixedLengthParamType (parameterID)</code>
<b>Parameters</b>	-> parameterID A UInt16 containing the parameter ID.
<b>Result</b>	Returns true if the parameter is a fixed length parameter type such as UInt32, or false if it is a variable length type.
<b>Compatibility</b>	Parameter IDs of this format are only used if <a href="#">Connection Manager Feature Set</a> is present.
<b>See Also</b>	<a href="#">CncProfileSettingSet</a> , <a href="#">CncProfileSettingGet</a>

## CncIsSystemFlags

<b>Purpose</b>	Macro that returns whether the parameter value is a system flag.
<b>Declared In</b>	ConnectionMgr.h
<b>Prototype</b>	<code>CncIsSystemFlags (parameterID)</code>
<b>Parameters</b>	-> parameterID The UInt16 containing the parameter ID.
<b>Result</b>	Returns true if the parameter type is a system flag. Returns false otherwise.
<b>Compatibility</b>	Parameter IDs of this format are only used if <a href="#">Connection Manager Feature Set</a> is present.
<b>See Also</b>	<a href="#">CncProfileSettingSet</a> , <a href="#">CncProfileSettingGet</a>

## CncIsSystemRange

<b>Purpose</b>	Macro that specifies whether the parameter is in the system range or in the third-party range.
<b>Declared In</b>	ConnectionMgr.h
<b>Prototype</b>	<code>CncIsSystemRange (parameterID)</code>
<b>Parameters</b>	-> parameterID A UInt16 containing the parameter ID.
<b>Result</b>	Returns true if the parameter ID is defined by Palm OS, or false if it is defined by a third party.
<b>Compatibility</b>	Parameter IDs of this format are only used if <a href="#">Connection Manager Feature Set</a> is present.
<b>See Also</b>	<a href="#">CncProfileSettingSet</a> , <a href="#">CncProfileSettingGet</a>

## CncIsThirdPartiesRange

<b>Purpose</b>	Macro that specifies whether the parameter is defined by a third party.
<b>Declared In</b>	ConnectionMgr.h
<b>Prototype</b>	<code>CncIsThirdPartiesRange (parameterID)</code>
<b>Parameters</b>	-> parameterID A UInt16 containing the parameter ID.
<b>Result</b>	Returns true if the parameter is a third-party parameter, or false if it is a system parameter.
<b>Compatibility</b>	Parameter IDs of this format are only used if <a href="#">Connection Manager Feature Set</a> is present.
<b>See Also</b>	<a href="#">CncProfileSettingSet</a> , <a href="#">CncProfileSettingGet</a>

## CnclsVariableLengthParamType

<b>Purpose</b>	Macro that returns whether the parameter value is a variable-length type.
<b>Declared In</b>	ConnectionMgr.h
<b>Prototype</b>	CncIsVariableLengthParamType (parameterID)
<b>Parameters</b>	-> parameterID A UInt16 containing the parameter ID.
<b>Result</b>	Returns true if the parameter is a variable-length string or a buffer or false if it holds a fixed-length type such as an integer.
<b>Compatibility</b>	Parameter IDs of this format are only used if <a href="#">Connection Manager Feature Set</a> is present.
<b>See Also</b>	<a href="#">CncProfileSettingSet</a> , <a href="#">CncProfileSettingGet</a>

## CncProfileCloseDB

<b>Purpose</b>	Closes the Connection Manager profile database.	
<b>Declared In</b>	ConnectionMgr.h	
<b>Prototype</b>	Err CncProfileCloseDB (void)	
<b>Parameters</b>	None.	
<b>Result</b>	errNone	No error.
	kCncErrDBAccessFailed	The database could not be closed or this is a reference counting error.
<b>Comments</b>	Use <a href="#">CncProfileOpenDB</a> and CncProfileCloseDB as an optimization if you make several Connection Manager calls in succession. All Connection Manager calls open the profile database	

when they begin and close the database when they are finished. The Connection Manager maintains a reference count that tells it whether the database is open. If you call CncProfileOpenDB before making another Connection Manager call, the next call does not open or close the database. This saves your application the overhead of opening and closing the database each time a call is made.

**Compatibility** Implemented only if [Connection Manager Feature Set](#) is present.

## CncProfileCount

**Purpose** Returns the number of connection profiles currently defined in the Connection Manager profile database.

**Declared In** ConnectionMgr.h

**Prototype** Err CncProfileCount (UInt16 \*profilesCountP)

**Parameters** <- profilesCountP  
The number of profiles.

**Result** errNone No error.

kCncErrDBAccessFailed  
The profile database could not be opened.

**Compatibility** Implemented only if [Connection Manager Feature Set](#) is present.

**See Also** [CncGetProfileList](#)

## CncProfileCreate

<b>Purpose</b>	Adds a profile record to the Connection Manager profile database.	
<b>Declared In</b>	ConnectionMgr.h	
<b>Prototype</b>	Err CncProfileCreate (CncProfileID *profileIdP)	
<b>Parameters</b>	<- profileIdP	Upon return, the unique ID of the new profile.
<b>Result</b>	errNone	No error.
	kCncErrDBAccessFailed	The profile database could not be opened.
	a Data Manager error	The new record could not be created.
<b>Comments</b>	This function creates a new empty record in the Connection Manager profile database. To populate the profile, use <a href="#">CncProfileSettingSet</a> to set parameter values, including the profile name. Use <a href="#">CncDefineParamID</a> if you need to store information unique to your profile.	
<b>Compatibility</b>	Implemented only if <a href="#">Connection Manager Feature Set</a> is present.	
<b>See Also</b>	<a href="#">CncAddProfile</a> , <a href="#">CncProfileDelete</a>	

## CncProfileDelete

<b>Purpose</b>	Deletes a profile.	
<b>Declared In</b>	ConnectionMgr.h	
<b>Prototype</b>	Err CncProfileDelete (CncProfileID profileId)	
<b>Parameters</b>	profileId	The ID of the profile to delete.
<b>Result</b>	errNone	No error.

kCncErrDBAccessFailed

The profile database could not be opened, or  
the record could not be deleted.

kCncErrProfileParamNotFound

The database does not contain a profile with the  
specified ID.

**Comments** The profiles that come preinstalled on the unit are read only and  
cannot be deleted.

**Compatibility** Implemented only if [Connection Manager Feature Set](#) is present.

**See Also** [CncDeleteProfile](#), [CncProfileCreate](#)

## CncProfileGetCurrent

**Purpose** Returns the ID of the currently selected profile in the Connection  
panel.

**Declared In** ConnectionMgr.h

**Prototype** Err CncProfileGetCurrent  
(CncProfileID \*profileIdP)

**Parameters** <- profileIdP The ID of the current profile.

**Result** errNone No error.

kCncErrDBAccessFailed  
The profile database could not be opened.

**Compatibility** Implemented only if [Connection Manager Feature Set](#) is present.

**See Also** [CncProfileGetIDFromIndex](#), [CncProfileGetIDFromName](#),  
[CncProfileGetIndex](#), [CncProfileSetCurrent](#)

## CncProfileGetIDFromIndex

<b>Purpose</b>	Returns the profile ID given its index into the Connection Manager profile database.						
<b>Declared In</b>	ConnectionMgr.h						
<b>Prototype</b>	<pre>Err CncProfileGetIDFromIndex (UInt16 index,                                CncProfileID *profileIdP)</pre>						
<b>Parameters</b>	<table><tr><td>-&gt; index</td><td>The index of the Connection Manager profile.</td></tr><tr><td>&lt;- profileIdP</td><td>The ID of the Connection Manager profile.</td></tr></table>	-> index	The index of the Connection Manager profile.	<- profileIdP	The ID of the Connection Manager profile.		
-> index	The index of the Connection Manager profile.						
<- profileIdP	The ID of the Connection Manager profile.						
<b>Result</b>	<table><tr><td>errNone</td><td>No error.</td></tr><tr><td>kCncErrDBAccessFailed</td><td>The profile database could not be opened.</td></tr><tr><td>kCncErrProfileParamNotFound</td><td>No profile at that index.</td></tr></table>	errNone	No error.	kCncErrDBAccessFailed	The profile database could not be opened.	kCncErrProfileParamNotFound	No profile at that index.
errNone	No error.						
kCncErrDBAccessFailed	The profile database could not be opened.						
kCncErrProfileParamNotFound	No profile at that index.						
<b>Compatibility</b>	Implemented only if <a href="#">Connection Manager Feature Set</a> is present.						
<b>See Also</b>	<a href="#">CncProfileGetIDFromName</a> , <a href="#">CncProfileGetCurrent</a> , <a href="#">CncProfileGetIndex</a>						

## CncProfileGetIDFromName

<b>Purpose</b>	Returns the profile ID given its name.	
<b>Declared In</b>	ConnectionMgr.h	
<b>Prototype</b>	<pre>Err CncProfileGetIDFromName (const Char *profileNameP, CncProfileID *profileIdP)</pre>	
<b>Parameters</b>	-> <code>profileNameP</code>	The name of the profile. The name is displayed in a pop-up list in the Connection panel. If you pass the string “-Current-”, this function returns the ID of the current profile.
	<- <code>profileIdP</code>	The profile ID.
<b>Result</b>	<code>errNone</code>	No error.
	<code>kCncErrDBAccessFailed</code>	The profile database could not be opened.
	<code>kCncErrProfileParamNotFound</code>	No profile with the specified name.
<b>Compatibility</b>	Implemented only if <a href="#">Connection Manager Feature Set</a> is present.	
<b>See Also</b>	<a href="#">CncProfileGetCurrent</a> , <a href="#">CncProfileGetIDFromIndex</a> , <a href="#">CncProfileGetIndex</a>	

## CncProfileGetIndex

<b>Purpose</b>	Returns the index of the profile given its ID.	
<b>Declared In</b>	ConnectionMgr.h	
<b>Prototype</b>	<pre>Err CncProfileGetIndex (CncProfileID profileId, UInt16 *indexP)</pre>	
<b>Parameters</b>	-> <code>profileId</code>	The profile ID.

## Connection Manager

### Connection Manager Functions

---

<- indexP      The index of the profile's record in the Connection Manager profile database.

**Result**    errNone      No error.

                  kCncErrDBAccessFailed  
                  The profile database could not be opened.

                  kCncErrProfileParamNotFound  
                  No profile with the specified ID.

**Compatibility**    Implemented only if [Connection Manager Feature Set](#) is present.

**See Also**    [CncProfileGetIDFromIndex](#)

## CncProfileOpenDB

**Purpose**    Opens the Connection Manager profile database.

**Declared In**    ConnectionMgr.h

**Prototype**    Err CncProfileOpenDB (void)

**Parameters**    None

**Result**    errNone      No error.

                  kCncErrDBAccessFailed  
                  The profile database could not be opened.

**Comments**    Use CncProfileOpenDB and [CncProfileCloseDB](#) as an optimization if you make several Connection Manager calls in succession. All Connection Manager calls open the profile database when they begin and close the database when they are finished. The Connection Manager maintains a reference count that tells it whether the database is open. If you call CncProfileOpenDB before making another Connection Manager call, the next call does not open or close the database. This saves your application the overhead of opening and closing the database each time a call is made.

The Connection Manager profile database is created if it does not exist.

**Compatibility** Implemented only if [Connection Manager Feature Set](#) is present.

## CncProfileSetCurrent

**Purpose** Sets the current profile.

**Declared In** ConnectionMgr.h

**Prototype** Err CncProfileSetCurrent (CncProfileID profileId)

**Parameters** -> profileId The ID of the profile to be made current.

**Result** errNone No error.

kCncErrDBAccessFailed  
The profile database could not be opened.

**Comments** The current profile is the profile that is used for the next network connection attempt.

**Compatibility** Implemented only if [Connection Manager Feature Set](#) is present.

**See Also** [CncProfileGetCurrent](#)

## CncProfileSettingGet

<b>Purpose</b>	Obtains a value stored in one of the Connection Manager profiles.	
<b>Declared In</b>	ConnectionMgr.h	
<b>Prototype</b>	<pre>Err CncProfileSettingGet (CncProfileID profileId,                            UInt16 paramId, void *paramBufferP,                            UInt16 *ioParamSizeP)</pre>	
<b>Parameters</b>	<code>-&gt; profileId</code>	The ID of the profile from which to obtain a parameter value.
	<code>-&gt; paramId</code>	The ID of the parameter to obtain. See “ <a href="#">Profile Parameter Constants</a> ” for a list of the parameters used in the profiles that come preinstalled on the device.
	<code>&lt;- paramBufferP</code>	A pointer to a buffer into which to write the parameter value. If the parameter stores a variable-sized value, you can determine the necessary size by passing NULL for paramBufferP. Upon return, paramSize contains the required size.
	<code>&lt;-&gt; ioParamSizeP</code>	On input, a pointer to the size of the buffer into which to write the parameter. On output, points to the number of bytes written to paramBufferP.
<b>Result</b>	<code>errNone</code>	No error.
	<code>kCncErrDBAccessFailed</code>	The profile database could not be opened.
	<code>kCncErrProfileGetParamFailed</code>	The Connection Manager failed to obtain the value of the parameter.
	<code>kCncErrProfileBadSystemFlagBitnum</code>	An attempt was made to obtain the value of a system flag that is undefined.

kCncErrProfileBadParamSize

The paramBufferP buffer is too small.  
ioParamSizeP contains the correct size for the  
buffer.

kCncErrProfileParamNotFound

The specified parameter is not defined in the  
profile.

**Compatibility** Implemented only if [Connection Manager Feature Set](#) is present.

**See Also** [CncProfileSettingSet](#), [CncGetProfileInfo](#)

## CncProfileSettingSet

**Purpose** Sets a parameter value in the specified profile.

**Declared In** ConnectionMgr.h

**Prototype** Err CncProfileSettingSet (CncProfileID iProfileId,  
                  UInt16 paramId, const void \*paramBufferP,  
                  UInt16 paramSize)

**Parameters**

- > iProfileId The ID of the profile.
- > paramId The ID of the parameter. See “[Profile Parameter Constants](#)” for a list of the parameters defined in the preinstalled connection profiles.
- > paramBufferP A pointer to the value to set for this parameter.
- > paramSize The size of the buffer passed in paramBufferP. See “[Profile Parameter Size Constants](#)”.

**Result** errNone No error.

kCncErrDBAccessFailed

The profile database could not be opened.

## **Connection Manager**

### *Connection Manager Functions*

---

kCncErrProfileParamNotFound

No profile with the specified ID.

kCncErrProfileSetParamFailed

The Connection Manager failed to set the value of the parameter.

kCncErrProfileBadParamSize

The paramBufferP buffer is too small. Most likely, the passed in size does not allow space for a string parameter's null terminator.

kCncErrProfileParamNameHasChange

An attempt was made to set the profile name to a name that is already used. The Connection Manager appends a duplication number to the end of the name and returns this error. You should use [CncProfileSettingGet](#) to find out the new name.

**Compatibility** Implemented only if [Connection Manager Feature Set](#) is present.

**See Also** [CncDefineParamID](#), [CncProfileSettingGet](#)

# Exchange Manager

---

This chapter describes the Exchange Manager API declared in the header file `ExgMgr.h` and the Exchange Local Library API declared in the header file `ExgLocalLib.h`. It discusses the following topics:

- [Exchange Manager Data Structures](#)
- [Exchange Manager Constants](#)
- [Exchange Manager Functions](#)
- [Application-Defined Functions](#)

For more information on the Exchange Manager, see the chapter “[Object Exchange](#)” on page 1 of *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## Exchange Manager Data Structures

### **ExgAskResultType**

The `ExgAskResultType` enum defines possible values for the `result` field of the [`sysAppLaunchCmdExgAskUser`](#) launch code parameter block.

```
typedef enum {
    exgAskDialog,
    exgAskOk,
    exgAskCancel }
ExgAskResultType;
```

### Value Descriptions

exgAskDialog	The Exchange Manager should display a dialog that prompts the user to confirm the receipt of data. See <a href="#">ExgDoDialog</a> .
exgAskOk	Accept the data.
exgAskCancel	Reject the data.

## ExgGoToType

The ExgGoToType structure defines the goToParams field of the [ExgSocketType](#) structure. Applications that want to be launched after the data is received place their creator IDs in the goToCreator field and define the goToParams field. The values in this structure are copied to the [sysAppLaunchCmdGoto](#) launch code's parameter block.

```
typedef struct {
    UInt16 dbCardNo;
    LocalID dbID;
    UInt16 recordNum;
    UInt32 uniqueID;
    UInt32 matchCustom;
} ExgGoToType;
```

### Field Descriptions

dbCardNo	The card number of the database that contains the added record.
dbID	The local ID of the database that contains the added record.
recordNum	The index of the record that was added.
uniqueID	The unique ID of the record that was added. This field is not used.
matchCustom	Application-specific information.

## **ExgLocalSocketInfoType**

The ExgLocalSocketInfoType structure identifies information specific to the Local Exchange Library. The socketRef field of the [ExgSocketType](#) structure is set to this structure when you send and receive data using the Local Exchange Library. The Local Exchange Library creates this structure if it does not already exist. You only need to create it if you want to supply non-default values for the noAsk or previewInfoP fields.

```
typedef struct {
    Boolean freeOnDisconnect;
    Boolean noAsk;
    ExgPreviewInfoType *previewInfoP;
    FileHand tempFileH;
    Err err;
    ExgLocalOpType op;
} ExgLocalSocketInfoType;
```

### **Field Descriptions**

**freeOnDisconnect** Whether the structure is freed when the [ExgDisconnect](#) call is made. The default is true. In general, code that allocates a structure should be responsible for freeing that structure. Therefore, if you have allocated ExgLocalSocketInfoType, you should set this field to false and explicitly free the structure when you are finished with it.

**noAsk** Set to true to disable the display of the exchange dialog. If you want to, for example, create a vCalendar object and send it to the Datebook application in response to a user command, you probably want to set noAsk to true so that the user does not have to confirm the receipt of the data they just requested you to send.

## **Exchange Manager**

### *Exchange Manager Data Structures*

---

previewInfoP	A pointer to an <a href="#">ExgPreviewInfoType</a> structure, used to display a preview of the data. If you wanted to simply use another application to help display data, you would create and initialize this structure.
tempFileH	A temporary buffer that the Local Exchange Library uses. Do not set this field directly; the Local Exchange Library should set it.
err	The error code returned from the Local Exchange Library. Do not set this field directly; the Local Exchange Library should set it.
op	The operation in progress. Do not set this field directly. The Local Exchange Library sets this field to one of the following:  exgLocalOpNone No operation in progress. exgLocalOpPut A send is in progress. exgLocalOpAccept A receive is in progress. exgLocalOpGet A get is in progress. exgLocalOpGetSender The library is receiving information from the sending application during a get operation.

## **ExgPreviewInfoType**

The ExgPreviewInfoType structure provides information to the [ExgNotifyPreview](#) function. The ExgNotifyPreview function uses this information to have the application display a preview of the data to be received in the exchange dialog.

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

### **Field Descriptions**

- > **version** Set this field to 0 to specify version 0 of this structure.
- > **socketP** A pointer to the socket structure (see [ExgSocketType](#)). The libraryRef field must identify the exchange library from which preview data should be received, and the target, type, or name field should be defined as well.
- > **op** One of the following constants:
  - exgPreviewDialog** Display a modal dialog containing the preview. This constant is only used in situations where one application launches another to display data.
  - exgPreviewDraw** The preview is a graphic.
  - exgPreviewLongString** The preview is a long string.

`exgPreviewQuery`

Ask the application which preview operations it supports. The answer is returned in the `types` field. If the application does not support any preview modes, the `error` field contains `exgErrNotSupported`.

`exgPreviewShortString`

The preview is a short string.

`<- string` A buffer into which the application places the string preview if `exgPreviewLongString` or `exgPreviewShortString` is specified.

`-> size` The allocated size of the `string` field.

`-> bounds` The bounds of the rectangle in which the application draws the graphic if `exgPreviewDraw` is specified.

`<- types` Upon return from an `exgPreviewQuery` operation, a bit field identifying the types of previews the application supports.

`<- error` The error code returned from the application. If `errNone`, the preview operation was successful.

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

**Compatibility** This structure is only defined if [4.0 New Feature Set](#) is present.

## **ExgSocketType**

The ExgSocketType structure defines an Exchange Manager socket, which is passed to most Exchange Manager functions. The ExgSocketPtr type points to a ExgSocketType structure.

```
typedef struct ExgSocketType {  
    UInt16 libraryRef;  
    UInt32 socketRef;  
    UInt32 target;  
    UInt32 count;  
    UInt32 length;  
    UInt32 time;  
    UInt32 appData;  
    UInt32 goToCreator;  
    ExgGoToType goToParams;  
    UInt16 localMode:1;  
    UInt16 packetMode:1;  
    UInt16 noGoTo:1;  
    UInt16 noStatus:1;  
    UInt16 preview:1;  
    UInt16 reserved:11;  
    Char *description;  
    Char *type;  
    Char *name;  
} ExgSocketType;  
  
typedef ExgSocketType* ExgSocketPtr;
```

Note that when data is received, some of the fields in this structure may not have values. When you are sending data, it is recommended that you provide values for all of these fields, but you should not rely on receiving values for the fields marked optional.

### Field Descriptions

libraryRef	The exchange library in use. When an application or library receives a socket, this field is already assigned.
	When sending data, applications may identify the exchange library they want to connect with by providing a URL in the name field. If so, they should use 0 for the libraryRef field. The Exchange Manager then determines which library corresponds to the URL and assigns the libraryRef field. See the Comments in the <a href="#">ExgPut</a> function description for more information.
socketRef	The connection identifier. This value is supplied by the exchange library when a connection is established. It contains any necessary library-specific data.
target	The creator ID of the application that should receive the message.
count	The number of objects in this connection, usually 1 (optional).
length	The total byte count for all objects being sent (optional).
time	The last modified time of object (optional).
appData	Application-specific information (optional).
goToCreator	The creator ID of the application to launch using the <a href="#">sysAppLaunchCmdGoto</a> launch code after the item is received if noGoTo is 0. The value is assigned by the application that receives the object. See the Comments section in <a href="#">ExgDisconnect</a> for more information.
goToParams	If goToCreator is specified, then this field contains data that is copied into the launch code's parameter block. See <a href="#">ExgGoToType</a> .

localMode	Set to 1 to exchange with local device only. A localMode of 1 is equivalent to specifying a URL with the <code>exgLocalPrefix</code> . Set to 0 to enable an exchange with a remote machine. The default is 0.
packetMode	Set to 1 to use connectionless packet mode (Ultra). The default is 0. Ultra mode is not currently supported.
noGoTo	Set to 1 to disable launching the application with <code>sysAppLaunchCmdGoto</code> . The default is 0.
noStatus	If true, the exchange library should not display a progress dialog. If false, the library can display a progress dialog. The default is false.  The Exchange Manager sets and clears this bit at various times while data is received. Applications may also want to set this bit if they use the Local Exchange Library and want to prevent the progress dialog from being displayed during a send.
preview	If true, a preview is in progress. The <code>ExgNotifyPreview</code> function sets this bit while the preview takes place and clears it when the preview is finished. Exchange libraries should not discard any data while a preview is in progress because the full data must be sent later if the receiving user accepts it.
reserved	Reserved system flags.
description	A pointer to the text description of the object (optional).
type	A pointer to the MIME type of the object (optional).

**name** The name of the object being sent. This can be a URL whose scheme identifies the exchange library to connect with.

If the name has a colon, it is treated as a URL.

**Compatibility** The noGoTo and noStatus flags are only defined if [3.5 New Feature Set](#) is present, and the noStatus flag has no effect unless [4.0 New Feature Set](#) is present. The preview flag is only defined if [4.0 New Feature Set](#) is present.

## Exchange Manager Constants

### Registry ID Constants

The registry ID constants are used in the Exchange Manager registry. Exchange libraries register for the URL prefixes they handle, and applications register for the types of data they receive. The registry ID constants specify which type of data is being registered for.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
exgRegCreatorID	0xffffb	Register for a creator ID. The target field of the <a href="#">ExgSocketType</a> contains the creator ID of the application that should receive the data. Typically, the application with the matching creator ID receives the data, but it is possible for one application to register for another's creator ID and receive data in its place.
exgRegSchemeID	0xffffc	Register for a URL scheme. Typically, only exchange libraries register for URL schemes. Applications can register for URL schemes, but they only receive the URL when <a href="#">ExgRequest</a> is called. If the name field of the ExgSocketType contains a colon (:), the portion of the URL before the colon is the URL scheme. The default library registered for URLs with that scheme will handle the message.
exgRegExtensionID	0xffffd	Register for a filename extension. If the name field of the ExgSocketType contains a period (.), the portion of the name after the last period is the filename extension. The application registered to handle files of that extension will handle the message.
exgRegTypeID	0xffffe	Register for a MIME type. If the type field of the ExgSocketType contains a value, the application registered to receive that MIME type handles the message.

**Compatibility** The exgRegCreatorID and exgRegSchemeID constants are only defined if [4.0 New Feature Set](#) is present.

## Predefined URL Schemes

The Exchange Manager provides these predefined URL schemes, for which exchange libraries can register.

## **Exchange Manager**

### *Exchange Manager Constants*

---

<b>Constant</b>	<b>Value</b>	<b>Description</b>
exgBeamScheme	"_beam"	The URL scheme for Beam commands. By default, the IR Library handles this scheme.
exgSendScheme	"_send"	The URL scheme for Send commands. The purpose of the Send command is to provide a choice of transport mechanisms to the user; therefore, any exchange library that sends data should register for this scheme.
exgLocalScheme	"_local"	The URL scheme for the Local Exchange Library.

**Compatibility** These constants are only defined if [4.0 New Feature Set](#) is present.

## **Predefined URL Prefixes**

The Exchange Manager provides the following prefixes, which can be used to construct URLs appropriate for the name field of the [ExgSocketType](#) structure. When sending data, applications provide a URL to identify the exchange library that should transport the data.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
exgBeamPrefix	(exgBeamScheme ":")	The URL to beam data.
exgSendPrefix	("?" exgSendScheme ":")	A URL for the general Send command. Because this URL begins with a question mark (?), the Exchange Manager displays a dialog with a list of exchange libraries registered for the exgSendScheme. The user then chooses the desired exchange library.

Constant	Value	Description
exgSendBeamPrefix	( "?" exgSendScheme ";" exgBeamScheme ":" )	A URL for the general Send command. The Exchange Manager displays a dialog with a list of exchange libraries registered for either the exgSendScheme or the exgBeamScheme.
exgLocalPrefix	( exgLocalscheme ":" )	The URL for using the Local Exchange Library.

**Compatibility** These constants are only defined if [4.0 New Feature Set](#) is present.

## Exchange Manager Functions

### ExgAccept

**Purpose** Accepts a connection from a remote device.

**Declared In** ExgMgr.h

**Prototype** Err ExgAccept (ExgSocketType \*socketP)

**Parameters** -> socketP A pointer to the socket structure (see [ExgSocketType](#)).

**Result** Returns one of the following error codes:

errNone	Success
---------	---------

exgErrBadLibrary	Couldn't find default exchange library
------------------	--

exgErrNotSupported	A preview is in progress, and the exchange library identified by libraryRef doesn't support preview mode
--------------------	--

Other error codes depend on the exchange library.  
Displays a fatal error message if socketP does not have a libraryRef specified.

**Comments** Applications call this function when launched with the [sysAppLaunchCmdExgReceiveData](#) or [sysAppLaunchCmdExgPreview](#) launch code. The launch code contains socketP in its parameter block. Applications should pass this socket to ExgAccept to accept the connection, then call [ExgReceive](#) one or more times to receive the data, and then call [ExgDisconnect](#) to disconnect.

---

**NOTE:** Don't create the socket on the receiving side of an exchange. The socket is passed to you in the command parameter block of the [sysAppLaunchCmdExgReceiveData](#) or [sysAppLaunchCmdExgPreview](#) launch code.

---

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present. Preview mode is supported only if [4.0 New Feature Set](#) is present.

**See Also** [ExgConnect](#), [ExgPut](#), [ExgGet](#)

## ExgConnect

**Purpose** Establishes a connection with a remote socket.

**Declared In** ExgMgr.h

**Prototype** Err ExgConnect (ExgSocketType \*socketP)

**Parameters** -> socketP A pointer to the socket structure (see [ExgSocketType](#)). Specify either a value for the libraryRef field or a URL in the name field. libraryRef should be 0 if the name field contains a URL.

**Result** Returns one of the following error codes:

errNone	Success
exgErrBadLibrary	Couldn't find exchange library
exgErrNotSupported	The library doesn't support the operation specified in socketP
exgErrUserCancel	The user cancelled the connection operation
exgMemError	There isn't enough free memory to respond to the request
exgErrNotEnoughPower	The battery does not have enough power to perform the operation

Other error codes depend on the exchange library.

#### **Comments**

Applications can call this function to initiate a connection for sending multiple objects or for performing two-way communications. Some exchange libraries support sending multiple objects but do not support this call. See “[Sending Multiple Objects](#)” on page 17 of *Palm OS Programmer’s Companion*, vol. II, *Communications* for more information.

Before calling this function, the application must initialize the socketP parameter. The socket should identify the exchange library to connect with by providing either a library reference number in the libraryRef field or a URL in the name field. The default exchange library registered for that type of URL handles the connection.

To provide users with a choice of transport mechanisms, specify a URL that begins with a question mark (?). The Exchange Manager displays a dialog with a list of all exchange libraries that respond to URLs of the specified type. If only one exchange library is registered for this URL scheme, no dialog is displayed.

For example, many applications on Palm OS® 4.0 or higher support a Send command. This command generates a URL with the prefix exgSendPrefix (see [Predefined URL Prefixes](#)). The Exchange Manager displays a dialog containing a list of libraries registered for that URL scheme. The user selects an exchange library, and that library’s ExgLibConnect function is called.

## **Exchange Manager**

### *Exchange Manager Functions*

---

If the library is not specified by either URL or library reference number (in the `lLibraryRef` field), the Exchange Manager by default uses the IR Library; however, if the `localMode` flag is set, the Local Exchange Library is used instead.

In addition to specifying the library, you can set the `count` field in `socketP` before making this call to indicate the number of objects that are going to be sent. Use a `count` of 0 if the number of objects isn't known in advance.

If no error is returned from `ExgConnect`, applications can follow this call either by sending multiple objects or requesting data from the remote device or both. To send an object, call [ExgPut](#) at the beginning of each object and call [ExgSend](#) one or more times per object to send the data. To request data from the remote device, use [ExgGet](#) (and then use [ExgReceive](#) to receive the requested data). You can use these calls in combination with each other to support two-way communications. After all of the objects have been sent and received, call [ExgDisconnect](#) to disconnect.

---

**IMPORTANT:** Not all exchange libraries support the sending of multiple objects or using `ExgGet` to request data.

---

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present. `ExgConnect` was for system use only until the release of Palm OS 4.0. Multiple object sending and identifying exchange libraries by URL are supported only if [4.0 New Feature Set](#) is present. On earlier releases, this function is an alias for the `ExgPut` function.

**See Also** [ExgPut](#), [ExgAccept](#), [ExgGet](#)

## ExgControl

**Purpose** Requests that an exchange library perform an operation.

**Declared In** ExgMgr.h

**Prototype** Err ExgControl (ExgSocketType \*socketP,  
UInt16 op, void \*valueP, UInt16 \*valueLenP)

<b>Parameters</b>	-> socketP	A pointer to the socket structure (see <a href="#">ExgSocketType</a> ). Specify either a value for the libraryRef field or a URL in the name field. libraryRef should be 0 if the name field contains a URL.
	-> op	A constant identifying the operation that the exchange library should perform. See the Comments section for more information.
	<-> valueP	Upon entry, a parameter that the exchange library requires to perform the operation, if any. Most operations do not require an input parameter. Upon return, contains the result of the operation.
	<-> valueLenP	The size of the valueP buffer. The size is updated upon return to show the actual length of the content returned.

**Result** Returns one of the following error codes:

errNone	Success
exgErrBadLibrary	Couldn't find the requested exchange library
exgErrNotSupported	The exchange library does not support the requested operation

Other error codes depend on the exchange library.

**Comments** The Exchange Manager uses this function to request information from the exchange library. Applications may also call this function. The Exchange Manager defines and uses a set of operation constants that it might send to any exchange library. These constants begin with the prefix `exgLibCtlGet`. The type of the variable pointed to by `valueP` depends on the type of operation to be performed. [Table 57.1](#) lists and describes the predefined Exchange Manager operations.

**Table 57.1 ExgControl operations for all exchange libraries**

Operation <code>exgLibCtlGet...</code>	value Data Type	Description
Preview	Boolean. Output only.	Returns <code>true</code> if the exchange library supports preview mode or <code>false</code> if not. If the exchange library does not respond to this operation, it is assumed to support preview mode.
Title	String buffer of size <code>exgTitleBufferSize</code> bytes. Output only.	Returns the name of the exchange library as it should appear in the Send dialog. All exchange libraries must respond to this operation.
Version	UInt16. Output only.	Returns the version of the exchange library API that this library implements. The constant <code>exgLibAPIVersion</code> defines the current version number. If the exchange library does not respond to this operation, the library supports the version of the Exchange Library API defined in Palm OS 4.0.

An exchange library may also define its own operations. For example, the IR Library supports operations to enable or disable beaming, to set the baud rates, or to use the serial port (see “[IR Control Constants](#)” on page 1378). The SMS Library supports operations that allow you to set the SMS preferences for sending messages or to manipulate multipart messages (see “[SMS Control](#)

[Constants](#)" on page 2232). Operations specific to an exchange library are numbered starting at `exgLibCtlSpecificOp`.

The `socketP` passed to this function must identify an exchange library either using the `libraryRef` field or using a URL in the `name` field. The Comments section in [ExgConnect](#) describes how an application should identify the exchange library.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## ExgDBRead

**Purpose** Converts a Palm OS database from its internal format and writes it to storage RAM.

**Declared In** `ExgMgr.h`

**Prototype** `Err ExgDBRead (ExgDBReadProcPtr readProcP,  
ExgDBDeleteProcPtr deleteProcP, void* userDataP,  
LocalID* dbIDP, UInt16 cardNo,  
Boolean* needResetP, Boolean keepDates)`

<b>Parameters</b>	-> <code>readProcP</code>	A pointer to a function that reads in the database and passes it to <code>ExgDBRead</code> . See <a href="#">ExgDBReadProcPtr</a> for details.
	-> <code>deleteProcP</code>	A pointer to a function that is called if a database with an identical name already exists on the device. See <a href="#">ExgDBDeleteProcPtr</a> for details.
	-> <code>userDataP</code>	A pointer to any data you want to pass to either the <code>readProcP</code> or <code>deleteProcP</code> functions. Often, this parameter is used to pass the <a href="#">ExgSocketType</a> that is required by many Exchange Manager functions.
	<- <code>dbIDP</code>	The ID of the database that <code>ExgDBRead</code> created on the local device.
	<- <code>cardNo</code>	The number of the card on which the database was stored by <code>ExgDBRead</code> .

## **Exchange Manager**

### *Exchange Manager Functions*

---

<- needResetP      Set to true by ExgDBRead if the dmHdrAttrResetAfterInstall attribute bit is set in the received database.

-> keepDates      Specify true to retain the creation, modification, and last backup dates as set in the received database header. Specify false to reset these dates to the current date.

**Result**      Returns errNone if successful; otherwise, returns one of the data manager error codes (dmErr...) or a callback-specific error code. (If the readProcP function returns an error, it is also returned by ExgDBRead.)

**Comments**      This function converts data received from an exchange library or from any other transport mechanism into a Palm OS database and stores that database in the storage heap. It is not required that you use this function in conjunction with Exchange Manager calls. That is, it's possible to use this function to perform other operations, such as converting a database created on the desktop computer to a Palm OS formatted database in the storage heap.

The primary use of this function, however, is to receive a database that has been beamed onto the device. In this case, call ExgDBRead in response to the launch code

[sysAppLaunchCmdExgReceiveData](#) after calling [ExgAccept](#) to accept the connection. Place the call to [ExgReceive](#) in the read callback function you passed as the readProcP parameter. Pass the [ExgSocketType](#) structure returned from ExgAccept in the userDataP parameter so that you have access to it in the read callback function.

The read callback function performs the actual reading of data. ExgDBRead calls the read callback function multiple times. Each time, the sizeP parameter contains the number of bytes ExgDBRead expects the data returned in dataP to contain. It's important for the read callback function to set the number of bytes (in sizeP) that it actually placed in dataP if it's not the same as what ExgDBRead expected. ExgDBRead stops calling the read callback function after 0 is returned in sizeP.

The callback function you pass in deleteProcP handles the case where the database being read already exists on the device. It is called only in that circumstance. The callback function may want to close the database if it is open, change the existing database's name, or delete the existing database to allow an overwrite. See [ExgDBDeleteProcPtr](#) for more information.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [ExgDBWrite](#)

## ExgDBWrite

<b>Purpose</b>	Converts a given Palm OS database from its internal format on the local device and writes it using a function you supply.										
<b>Declared In</b>	<code>ExgMgr.h</code>										
<b>Prototype</b>	<pre>Err ExgDBWrite (ExgDBWriteProcPtr writeProcP, void* userDataP, const char* nameP, LocalID dbID, UInt16 cardNo)</pre>										
<b>Parameters</b>	<table><tr><td>-&gt; writeProcP</td><td>A pointer to a function that writes out the database identified by <code>dbID</code>. See <a href="#">ExgDBWriteProcPtr</a> for details.</td></tr><tr><td>-&gt; userDataP</td><td>A pointer to any data you want to pass to the <code>writeProcP</code> function. Often, this parameter is used to pass the <a href="#">ExgSocketType</a> that is required by many Exchange Manager functions.</td></tr><tr><td>-&gt; nameP</td><td>A pointer to the name of the database that you want <code>ExgDBWrite</code> to write. This database is passed to <code>writeProcP</code>.</td></tr><tr><td>-&gt; dbID</td><td>The ID of the database that you want <code>ExgDBWrite</code> to pass to <code>writeProcP</code>. If you don't supply an ID, then <code>nameP</code> is used to search for the database by name.</td></tr><tr><td>-&gt; cardNo</td><td>The number of the card on which to look for the database identified by <code>nameP</code>.</td></tr></table>	-> writeProcP	A pointer to a function that writes out the database identified by <code>dbID</code> . See <a href="#">ExgDBWriteProcPtr</a> for details.	-> userDataP	A pointer to any data you want to pass to the <code>writeProcP</code> function. Often, this parameter is used to pass the <a href="#">ExgSocketType</a> that is required by many Exchange Manager functions.	-> nameP	A pointer to the name of the database that you want <code>ExgDBWrite</code> to write. This database is passed to <code>writeProcP</code> .	-> dbID	The ID of the database that you want <code>ExgDBWrite</code> to pass to <code>writeProcP</code> . If you don't supply an ID, then <code>nameP</code> is used to search for the database by name.	-> cardNo	The number of the card on which to look for the database identified by <code>nameP</code> .
-> writeProcP	A pointer to a function that writes out the database identified by <code>dbID</code> . See <a href="#">ExgDBWriteProcPtr</a> for details.										
-> userDataP	A pointer to any data you want to pass to the <code>writeProcP</code> function. Often, this parameter is used to pass the <a href="#">ExgSocketType</a> that is required by many Exchange Manager functions.										
-> nameP	A pointer to the name of the database that you want <code>ExgDBWrite</code> to write. This database is passed to <code>writeProcP</code> .										
-> dbID	The ID of the database that you want <code>ExgDBWrite</code> to pass to <code>writeProcP</code> . If you don't supply an ID, then <code>nameP</code> is used to search for the database by name.										
-> cardNo	The number of the card on which to look for the database identified by <code>nameP</code> .										
<b>Result</b>	Returns <code>errNone</code> if successful; otherwise, returns one of the data manager error codes ( <code>dmErr...</code> ) or a callback-specific error code. (If the <code>writeProcP</code> function returns an error, it is also returned by <code>ExgDBWrite</code> .)										
<b>Comments</b>	This function converts a Palm OS formatted database on the storage heap into a stream of bytes that can be sent over the Internet or over										

any other transport mechanism. It is not required that you use this function in conjunction with Exchange Manager calls.

The primary use of this function, however, is to write a database that is going to be beamed onto another device. In this case, call ExgDBWrite after establishing the connection with [ExgPut](#). Place the call to [ExgSend](#) in the write callback function you passed as the writeProcP parameter. Pass the [ExgSocketType](#) structure returned from ExgSend in the userDataP parameter so that you have access to it in the write callback function.

The write callback function performs the actual writing of data. ExgDBWrite calls the write callback function multiple times. Each time, the sizeP parameter contains the number of bytes of dataP that are to be written. If the write callback function didn't handle it all, it's important that it set in sizeP the number of bytes that it did handle successfully. ExgDBWrite stops calling the write callback function after 0 is returned in sizeP.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

**See Also** [ExgDBRead](#)

## ExgDisconnect

**Purpose** Terminates an Exchange Manager transfer and disconnects.

**Declared In** ExgMgr.h

**Prototype** Err ExgDisconnect (ExgSocketType \*socketP,  
Err error)

**Parameters** -> socketP A pointer to the socket structure (see [ExgSocketType](#)) identifying the connection to terminate.

## Exchange Manager

### Exchange Manager Functions

---

-> error Any error that occurred. This parameter tells the exchange library why the connection is being broken. Normally the error code from [ExgSend](#) or [ExgReceive](#) is passed in here.

**Result** Returns one of the following error codes:

errNone	Success
exgErrBadLibrary	Couldn't find default exchange library
exgMemError	Couldn't read data to send
exgErrUserCancel	User cancelled transfer

Other error codes depend on the exchange library.

May display a fatal error message if `socketP` doesn't contain a `libraryRef` value.

**Comments** Applications must call this function when finished sending data or receiving data. It terminates the connection made with [ExgConnect](#), [ExgAccept](#), [ExgPut](#), or [ExgGet](#).

In the `error` parameter, pass any error that occurs during the application loop, including errors returned from other Exchange Manager functions. This ensures that the connection is shut down knowing that it failed rather than succeeded.

It's especially important to check the result code from this function, since this will tell you if the transfer was successful. An `errNone` return value means that the item was delivered to the destination successfully. It does not mean that the user on the other end actually kept the data.

`ExgDisconnect` is used after sending and receiving. When receiving, the application can insert its creator ID into the `goToCreator` field in the socket structure and add other goto information in the `goToParams` field. After the application returns from the [sysAppLaunchCmdExgReceiveData](#) launch code, the exchange library may call [ExgNotifyGoto](#), which launches the `goToCreator` application with the standard launch code [sysAppLaunchCmdGoto](#).

**IMPORTANT:** Placing your creator ID in the `goToCreator` field is no longer a guarantee that you receive this launch code starting in Palm OS 4.0 because Palm OS 4.0 supports the sending of multiple objects at once. Thus, another application might overwrite the `goToCreator` field after your application has disconnected, making that application the recipient of the launch code.

---

Note that some exchange libraries wait to establish a connection until `ExgDisconnect` is called. The IR Library, for example, buffers the data that it receives and then waits until `ExgDisconnect` to actually send this data unless `ExgConnect` is called to establish a multi-object send connection.

**Compatibility**

Implemented only if [3.0 New Feature Set](#) is present.

Prior to Palm OS release 4.0, the Exchange Manager always launched the `goToCreator` application, if one was provided, upon return from this function. If [4.0 New Feature Set](#) is present, the Exchange Manager does not launch the `goToCreator` application. Exchange libraries that want the previous behavior must explicitly call [ExgNotifyGoto](#).

**See Also**

[ExgReceive](#), [ExgSend](#)

## ExgDoDialog

<b>Purpose</b>	Displays a dialog that allows users to accept or reject the receipt of data.	
<b>Declared In</b>	ExgMgr.h	
<b>Prototype</b>	Boolean ExgDoDialog (ExgSocketType *socketP, ExgDialogInfoType *infoP, Err *errP)	
<b>Parameters</b>	-> socketP	A pointer to the socket structure (see <a href="#">ExgSocketType</a> ) identifying the connection. A value must be provided for the libraryRef field.  Applications can obtain the socket structure from the <a href="#">sysAppLaunchCmdExgAskUser</a> launch code parameter block.
	<-> infoP	A pointer to an ExgDialogInfoType structure (see the Comments section below).
	<- errP	errNone if no error, or the error code if an error occurred. Currently, no errors are returned.
<b>Result</b>	Returns true if the user clicks the OK button on the dialog, or false otherwise.	
<b>Comments</b>	This function displays the exchange dialog, which prompts the user to accept or reject incoming data.  By default, the Exchange Manager calls this function if the receiving application doesn't handle the <a href="#">sysAppLaunchCmdExgAskUser</a> launch code or if it returns exgAskDialog from the launch code handler. When the Exchange Manager calls ExgDoDialog, the dialog displays a message similar to "Do you want to accept 'John Doe' into Address Book?" and allows the user to accept or reject the data. If the user clicks OK, the data should be received as an unfiled record.	

The Exchange Manager attempts to display a preview of the data in the exchange dialog to provide users with enough information to determine if they want to accept or reject the data. To display the preview data, it calls [ExgNotifyPreview](#). Applications wishing to support preview mode should respond to the launch code [sysAppLaunchCmdExgPreview](#). See the [ExgNotifyPreview](#) function's description for more information.

Applications may also want to allow users to select a category in which to accept the incoming data. To do so, handle [sysAppLaunchCmdExgAskUser](#) to call [ExgDoDialog](#) directly and pass it a pointer to an [ExgDialogInfoType](#) structure. The [ExgDialogInfoType](#) structure is defined as follows:

```
typedef struct {
    UInt16      version;
    DmOpenRef   db;
    UInt16      categoryIndex;
} ExgDialogInfoType;

-> version      Set this field to 0 to specify version 0 of this
                  structure.

-> db           A pointer to an open database that defines the
                  categories the dialog should display.

<- categoryIndex
                  The index of the category in which the user
                  wants to file the incoming data.
```

If db is valid, the function extracts the category information from the specified database and displays it in a pop-up list. Upon return, the categoryIndex field contains the index of the category the user selected, or dmUnfiledCategory if the user did not select a category.

If the call to [ExgDoDialog](#) is successful, your application is responsible for retaining the value returned in categoryIndex and using it to file the incoming data as a record in that category. One way to do this is to store the categoryIndex in the socket's appData field (see [ExgSocketType](#)) and then extract it from the socket in your response to the launch code [sysAppLaunchCmdExgReceiveData](#). For example:

## Exchange Manager

### Exchange Manager Functions

---

```
if (cmd == sysAppLaunchCmdExgReceiveData) {
    UInt16 category =
        (ExgSocketPtr)cmdPBP->appData;
    /* other declarations */

    /* Receive the data, and create a new record
       using the received data. indexNew is the
       index of this record. */

    if (category != dmUnfiledCategory) {
        UInt16 attr;
        Err err;
        err = DmRecordInfo(dbP, indexNew, &attr,
                           NULL, NULL);

        // Set the category to the one the user
        // specified, and mark the record dirty.
        if ((attr & dmRecAttrCategoryMask) !=
            category) {
            attr &= ~dmRecAttrCategoryMask;
            attr |= category | dmRecAttrDirty;
            err = DmSetRecordInfo(dbP, indexNew,
                                  &attr, NULL);
        }
    }
}
```

Some of the Palm OS built-in applications (Address Book, Memo, and ToDo) use this method of setting the category on data received through beaming. Refer to the example code for these applications provided in the SDK for a more complete example of how to use ExgDoDialog.

When you explicitly call ExgDoDialog, you must set the result field of the sysAppLaunchCmdExgAskUser launch code's parameter block to either exgAskOk (upon success) or exgAskCancel (upon failure) to prevent the system from displaying the dialog a second time.

**Compatibility** Implemented only if [3.5 New Feature Set](#) is present.

Preview mode display in the exchange dialog is implemented only if [4.0 New Feature Set](#) is present.

## ExgGet

**Purpose** Establishes a connection and requests an object from a remote device.

**Declared In** ExgMgr.h

**Prototype** Err ExgGet (ExgSocketType \*socketP)

**Parameters** -> socketP A pointer to the socket structure (see [ExgSocketType](#)). Specify either a value for the libraryRef field or a URL in the name field. libraryRef should be 0 if the name field contains a URL. The target, type, or name fields should identify the data being requested.

**Result** Returns one of the following error codes:

errNone	Success
exgErrBadLibrary	Couldn't find default exchange library
exgErrUserCancel	The user cancelled the operation
exgMemError	There is not enough free memory to perform the operation

Other error codes depend on the exchange library.

**Comments** Applications use this function to request data (initiate a send) from a remote device. Not all exchange libraries support this operation. Before calling this function, the application must initialize the socketP parameter. The socket should identify the exchange library to connect with by providing either a library reference number in the libraryRef field or a URL in the name field. The default exchange library registered for the URL's scheme handles

## **Exchange Manager**

### *Exchange Manager Functions*

---

the connection. The socket should also specify what data it is requesting by providing values for at least one of the target, name, and type fields. Specifying the data in the name field is the most common method.

To provide users with a choice of transport mechanisms, the application can provide a URL that begins with a question mark (?). The Exchange Manager displays a dialog with a list of all exchange libraries that respond to URLs of the specified type. If only one exchange library is registered for this URL scheme, no dialog is displayed.

If the library is not specified by either URL or library reference number, the Exchange Manager by default uses the IR Library; however, if the localMode flag is set, the Local Exchange Library is used instead.

Applications can use ExgGet to initiate a send from the Local Exchange Library. For more information, see “[Sending and Receiving Locally](#)” on page 32 of the *Palm OS Programmer’s Companion*, vol. II, *Communications*.

If no error is returned, applications should follow this call with one or more calls to [ExgReceive](#), to receive the data, or [ExgDisconnect](#), to disconnect.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [ExgPut](#), [ExgConnect](#)

## ExgGetDefaultApplication

<b>Purpose</b>	Retrieves the default application for the specified type of data or the default exchange library for URLs with the specified scheme.						
<b>Declared In</b>	ExgMgr.h						
<b>Prototype</b>	Err ExgGetDefaultApplication (UInt32 *creatorIDP, UInt16 id, const Char *dataTypeP)						
<b>Parameters</b>	<table><tr><td>&lt;- creatorIDP</td><td>A pointer to the creator ID of the default application or default exchange library.</td></tr><tr><td>-&gt; id</td><td>The registry ID constant identifying the type of data in dataTypeP. See <a href="#">Registry ID Constants</a>.</td></tr><tr><td>-&gt; dataTypeP</td><td>A pointer to a string that contains the type of data for which to retrieve the default application or library. If dataTypeP is a file extension, do not include the period (.). If it is a URL, do not include the colon (:).</td></tr></table>	<- creatorIDP	A pointer to the creator ID of the default application or default exchange library.	-> id	The registry ID constant identifying the type of data in dataTypeP. See <a href="#">Registry ID Constants</a> .	-> dataTypeP	A pointer to a string that contains the type of data for which to retrieve the default application or library. If dataTypeP is a file extension, do not include the period (.). If it is a URL, do not include the colon (:).
<- creatorIDP	A pointer to the creator ID of the default application or default exchange library.						
-> id	The registry ID constant identifying the type of data in dataTypeP. See <a href="#">Registry ID Constants</a> .						
-> dataTypeP	A pointer to a string that contains the type of data for which to retrieve the default application or library. If dataTypeP is a file extension, do not include the period (.). If it is a URL, do not include the colon (:).						
<b>Result</b>	Returns errNone if a match was found or exgErrNoKnownTarget if there is no default application or library for this type of data.						
<b>Comments</b>	You might use this function to see which application on this device will receive a particular type of data or to see which library on this device handles URLs of a particular scheme.  For example, to find out which application receives TXT files on this device, do the following:						
	<pre>UInt32 creatorID; Err error; error = ExgGetDefaultApplication(&amp;creatorID,                   exgRegExtensionID, "TXT"); if (!error) {     //creatorID contains default application.</pre>						
	To find out which exchange library handles URLs that use the beam prefix, do the following:						

```
UInt32 creatorID;
Err error;
error = ExgGetDefaultApplication(&creatorID,
    exgRegSchemeID, exgBeamScheme);
if (!error) {
    //creatorID contains default library.
```

It's possible to have several applications registered to receive the same type of data, but none of them is the default. When the Exchange Manager receives an object of that type, it selects an application to receive the data, and it selects that same application every time. The selected application effectively becomes the default for the data type even though it is not explicitly set as the default. If this is the case, the `ExgGetDefaultApplication` function returns the creator ID of this de-facto default application.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [ExgGetRegisteredApplications](#),  
[ExgGetRegisteredTypes](#), [ExgRegisterDatatype](#),  
[ExgSetDefaultApplication](#)

## ExgGetRegisteredApplications

**Purpose** Retrieves a list of all applications registered to receive data of a specified type.

**Declared In** ExgMgr.h

**Prototype** Err ExgGetRegisteredApplications  
(UInt32 \*\*creatorIDs\_P, UInt32 \*numApps\_P,  
Char \*\*names\_P, Char \*\*descriptions\_P, UInt16 id,  
const Char \*dataType\_P)

**Parameters** <- creatorIDs\_P An array of the creator IDs of the applications registered to receive objects of this type. Pass NULL for this parameter if you only want to know how many applications are registered for this type.

<- numAppsP	The number of applications registered to receive objects of this type. This is the number of elements in the creatorIDsP array, the namesP array, and the descriptionsP array.
<- namesP	A packed list of strings, suitable for passing to <a href="#">SysFormPointerArrayToStrings</a> , containing the names of the applications or libraries. Each string is no more than exgMaxTitleLen characters. Pass NULL for this parameter if you don't want to retrieve it.
<- descriptionsP	A packed list of strings, suitable for passing to <a href="#">SysFormPointerArrayToStrings</a> , containing the descriptions of the applications or libraries. Descriptions are specified when the applications or libraries register for data. Each string is no more than exgMaxDescriptionLength characters. Pass NULL for this parameter if you don't want to retrieve it.
-> id	The registry ID constant identifying the type of data in dataTypeP. See <a href="#">Registry ID Constants</a> .
-> dataTypeP	A pointer to a tab-delimited, null-terminated string listing the items to register. (Use \t for the tab character.) Each item in the string must be no more than exgMaxTypeLength characters. There can be no more than 16 types total.

**Result** Returns errNone upon success or exgMemError if the function cannot allocate space for the creator IDs, names, or descriptions.

---

**IMPORTANT:** This function allocates enough space for the creatorIDsP, namesP, and descriptionsP arrays as long as you do not pass NULL for the parameters. You are still responsible for freeing these arrays.

---

<b>Comments</b>	You might use this function to see which applications on this device can receive a particular type of data or to see which libraries on this device handle URLs of a particular scheme. You can also use it to build a list of choices from which the user can select a default application or default exchange library for a particular data type or URL scheme. For example, iMessenger uses this function to build a list of mailto handlers so that the user can choose one of them to be the default.  The Exchange Manager itself uses <code>ExgGetRegisteredApplications</code> to find exchange libraries when it is given a URL that begins with a question mark (?). It displays the returned list to the user in the Send With dialog.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">ExgGetDefaultApplication</a> , <a href="#">ExgGetRegisteredTypes</a> , <a href="#">ExgRegisterDatatype</a> , <a href="#">ExgSetDefaultApplication</a>

## ExgGetRegisteredTypes

<b>Purpose</b>	Retrieve a list of all data types for which a registration exists.				
<b>Declared In</b>	<code>ExgMgr.h</code>				
<b>Prototype</b>	<code>Err ExgGetRegisteredTypes (Char **dataTypesP,                   UInt32 *sizeP, UInt16 id)</code>				
<b>Parameters</b>	<table><tr><td><code>&lt;- dataTypesP</code></td><td>A packed list of strings, suitable for passing to <a href="#">SysFormPointerArrayToStrings</a>, containing a sorted list of data types for which a registration exists. Each string is no more than <code>exgMaxTypeLength</code> characters.</td></tr><tr><td><code>&lt;- sizeP</code></td><td>The number of elements in the <code>dataTypesP</code> array.</td></tr></table>	<code>&lt;- dataTypesP</code>	A packed list of strings, suitable for passing to <a href="#">SysFormPointerArrayToStrings</a> , containing a sorted list of data types for which a registration exists. Each string is no more than <code>exgMaxTypeLength</code> characters.	<code>&lt;- sizeP</code>	The number of elements in the <code>dataTypesP</code> array.
<code>&lt;- dataTypesP</code>	A packed list of strings, suitable for passing to <a href="#">SysFormPointerArrayToStrings</a> , containing a sorted list of data types for which a registration exists. Each string is no more than <code>exgMaxTypeLength</code> characters.				
<code>&lt;- sizeP</code>	The number of elements in the <code>dataTypesP</code> array.				

-> id      The type of data to search for. For example, you can search for all registered creator IDs, all registered MIME types, and so on.

**Result**    Returns errNone upon success or exgMemError if the function cannot allocate space for the data types array.

---

**IMPORTANT:** This function allocates enough space for the dataTypesP array as long as you do not pass NULL for the parameter. You are still responsible for freeing this array.

---

**Comments**    This function could be used to create an application that allows users to choose the default application for each data type.

**Compatibility**    Implemented only if [4.0 New Feature Set](#) is present.

**See Also**    [ExgGetDefaultApplication](#),  
[ExgGetRegisteredApplications](#), [ExgRegisterDatatype](#),  
[ExgSetDefaultApplication](#)

## ExgGetTargetApplication

**Purpose**    Retrieves the application that should receive a specific message. This function does not search for libraries.

**Declared In**    ExgMgr.h

**Prototype**    Err ExgGetTargetApplication  
(ExgSocketType \*socketP, Boolean unwrap,  
UIInt32 \*creatorIDP, Char \*descriptionP,  
UIInt32 descriptionSize)

**Parameters**    -> socketP      A pointer to the socket structure (see [ExgSocketType](#)). The structure should contain values for the target, type, or name fields.

## **Exchange Manager**

### *Exchange Manager Functions*

---

-> unwrap	If true, only an application that registered to receive the data type with the exgUnwrap flag set should be the target application. If false, the target application should be an application that registered with the exgUnwrap flag clear.
<- creatorIDP	The creator ID of the application that should receive this object.
<-> descriptionP	The application's description from the registry, if any.
-> descriptionSize	The size of the descriptionP buffer.

**Result** Returns one of the following error codes:

errNone	Success
exgErrTargetMissing	The target field contains a creator ID, but the application with that creator ID does not exist
exgErrNoKnownTarget	No application is registered to receive the data type

#### **Comments**

The Exchange Manager uses this function to determine which application should be launched to receive incoming data. Applications and libraries may call this function as well. ExgGetTargetApplication determines the target application by doing the following:

- If the socketP->target field contains a creator ID, the Exchange Manager searches the registry to see if an application is registered for that creator ID as the default application. If the registry does not contain an entry for the creator ID, it checks to see if the application identified by the creator ID is installed on this device. If an application is found for the target, that is the application returned.
- If the socketP->type field contains a MIME type, the Exchange Manager searches the registry for an application

registered to receive objects of that type. If one is found, that is the application returned.

- If the `socketP->name` field contains a period (.), the portion after the last period is taken to be the file extension. The Exchange Manager searches the registry for an application registered to receive a file with the specified extension. If one is found, that is the application returned. If not, `exgErrNoKnownTarget` is returned.

If more than one application is registered for the target, type, or file extension, this function returns the one that is registered as the default. If no application is registered as the default, then a specific application is chosen. The Exchange Manager chooses this same application each time. That is, each time a file with a TXT extension is sent with no target or MIME type specified, the `ExgGetTargetApplication` returns the same application to handle the receipt.

Set the `unwrap` parameter to `true` if the object was sent as part of another object, such as a vStock object that was sent as an attachment to an e-mail message. In this case, the Exchange Manager searches for an application that registered to receive the target, the type, or the file extension of the vStock object with the `exgUnwrap` flag set. If an application is found, the vStock object is delivered, and the exchange library should discard the object that contained it (the e-mail message). If there is no application registered to receive the data with the `exgUnwrap` flag set, this function returns `exgErrNoKnownTarget`. In this case, the exchange library should call `ExgNotifyReceive` again passing the entire e-mail message instead of just the vStock attachment.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [ExgSetDefaultApplication](#), [ExgNotifyPreview](#),  
[ExgNotifyReceive](#), [ExgRegisterDatatype](#)

## ExgNotifyGoto

<b>Purpose</b>	Launches the target application using <a href="#">sysAppLaunchCmdGoto</a> .	
<b>Declared In</b>	ExgMgr.h	
<b>Prototype</b>	Err ExgNotifyGoto (ExgSocketType *socketP, UInt16 flags)	
<b>Parameters</b>	-> socketP	A socket identifying the object to deliver (see <a href="#">ExgSocketType</a> ). The goToCreator field contains the application to be launched, and the goToParams field contains data for the launch code's parameter block.
	-> flags	Not currently used. Pass 0 for this parameter.
<b>Result</b>	Returns one of the following error codes:	
	errNone	Success or the goToCreator field is empty
	dmErr... (one of the data manager error codes)	The specified application could not be found
	memErrNotEnoughSpace	Not enough memory available to create the launch code's parameter block
<b>Comments</b>	<p>Exchange libraries call this function if they want to support immediate display of the received object. Applications do not call this function.</p> <p>Most exchange libraries should call ExgNotifyGoto after the return from <a href="#">ExgNotifyReceive</a> so that the user can inspect the newly received data. If the exchange library is most often used by a single application that does not require the launch code, this call to ExgNotifyGoto can be skipped. For example, the SMS Library does not call ExgNotifyGoto. SMS messages are received by the SMS Messenger application, which does not launch upon receiving data.</p>	

`ExgNotifyGoto` only launches an application if one is specified in the `goToCreator` field and the `noGoTo` parameter is `false`. If a `goToCreator` is not specified, it is not considered an error. This gives the application a way to override the default behavior.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [ExgNotifyReceive](#), [ExgDisconnect](#)

## ExgNotifyPreview

**Purpose** Displays a preview in the exchange dialog of the data to be received.

**Declared In** `ExgMgr.h`

**Prototype** `Err ExgNotifyPreview (ExgPreviewInfoType *infoP)`

**Parameters** `<-> infoP` An [ExgPreviewInfoType](#) structure containing information about the preview operation.

**Result** Returns one of the following error codes:

`errNone` Success

`exgErrNotSupported` The exchange library doesn't support preview mode

`exgErrNoKnownTarget` There is no application registered to receive the type of object

Other error codes depend on the application.

**Comments** This function performs the preview operation specified in the `op` field of the `infoP` parameter. The [ExgDoDialog](#) function calls this function to show a data preview in the exchange dialog. Exchange libraries might want to call this function in certain circumstances. An application rarely calls this function, but it may do so if it displays its own dialog in response to the launch code [sysAppLaunchCmdExgAskUser](#).

ExgNotifyPreview uses [ExgGetTargetApplication](#) to determine the appropriate target application for this data and then launches that application with the launch code [sysAppLaunchCmdExgPreview](#), passing infoP as the parameter block. The application responds to this launch code by accepting the connection, receiving the data from the exchange library, and depending on the operation requested, drawing the data into the infoP->bounds rectangle or returning it in the infoP->string field, and then disconnecting. The ExgDoDialog function uses the returned information to draw the preview portion of the dialog.

If the preview data is a string, the ExgNotifyPreview provides a series of fallback strings that are used if the exchange library doesn't support preview or the application doesn't respond to the launch code. If the application fails to return a string, this function provides one of the following:

- the data's description from socketP->description
- the filename in socketP->name
- the target application's description as stored in the exchange registry
- the MIME type in socketP->type
- the file extension in socketP->name

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [ExgNotifyReceive](#), [ExgDisconnect](#)

## ExgNotifyReceive

**Purpose** Delivers an object to the appropriate application using the registry.

**Declared In** ExgMgr.h

**Prototype** Err ExgNotifyReceive (ExgSocketType \*socketP,  
                  UInt16 flags)

**Parameters** <-> socketP      A pointer to the socket structure (see [ExgSocketType](#)).

-> flags	A bit field. Pass 0 or a combination of the following constants (OR the constants together to specify more than one):  exgUnwrap The object being delivered should only be handled by an application that registered to receive it with the exgUnwrap flag set.  exgNoAsk Do not ask the user to confirm receipt of data. If this constant is passed, the target application does not receive the <a href="#">sysAppLaunchCmdExgAskUser</a> launch code, and the Exchange Manager does not call <a href="#">ExgDoDialog</a> to display the user confirmation dialog.  exgGet Specifies that this is a request for the application to send data rather than to receive data.
----------	--

**Result** Returns one of the following error codes:

errNone	Success
exgErrTargetMissing	The target field contains a creator ID, but the application with that creator ID does not exist
exgErrNoKnownTarget	No application is registered to receive the data type
exgErrUserCancel	The user cancelled the operation
Other error codes depend on the application that is launched.	

**Comments** Exchange libraries call this function to initiate a receive operation on the receiving device. Applications do not call this function.  
The ExgNotifyReceive function uses [ExgGetTargetApplication](#) to determine which application

should receive the data, then sends that application the appropriate launch codes.

If the `flags` parameter is 0, a receive operation is assumed. The `ExgNotifyReceive` function does the following:

1. It sends the application the [`sysAppLaunchCmdExgAskUser`](#) launch code.
2. If the application returns `exgAskDialog` or does not respond to the launch code, it calls [`ExgDoDialog`](#), which sends the application the [`sysAppLaunchCmdExgPreview`](#) launch code to have the application receive preview data for the dialog.
3. It sends the application the [`sysAppLaunchCmdExgReceiveData`](#) launch code to tell the application to receive the data.

If the `flags` field contains the `exgNoAsk` flag, the first and second steps are skipped.

If the `flags` field contains `exgGet`, this function is a request for data to send to the remote device, not a request to receive data from the remote device. In this case, `ExgNotifyReceive` launches the target application with the [`sysAppLaunchCmdExgGetData`](#) launch code.

If the `flags` field has the `exgUnwrap` bit set, it means that the object to be received was sent as part of another object, and it should only be sent to an application that registered to receive it with the `exgUnwrap` flag set. For example, if the exchange library receives an e-mail message with an attached vStock object, the exchange library may call `ExgNotifyReceive` with the `exgUnwrap` flag set and a socket that describes the vStock data type to see if there is an application that registered to receive it directly. If no application is registered to receive vStock objects with the `exgUnwrap` flag set, `ExgNotifyReceive` returns `exgErrNoKnownTarget`. The exchange library should then call `ExgNotifyReceive` again, but this time without the `exgUnwrap` flag and with a socket that describes the e-mail message data type. This second call sends the object to the application registered to receive the e-mail message rather than its vStock attachment. That application may extract the vStock attachment from the message and use the Local Exchange Library to send it to an application registered to receive vStock objects normally (without the `exgUnwrap` flag).

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.  
ExgNotifyReceive was a system use only function until the release of Palm OS 4.0.

If the [4.0 New Feature Set](#) is **not** present, the flags parameter is not supported, so libraries cannot suppress the exchange dialog, send objects with attachments, or perform a get operation. These features are all added in the [4.0 New Feature Set](#). Also, if the 4.0 new feature set is not present, this function performs the equivalent of [ExgNotifyGoto](#) after the application has returned from receiving data. Exchange libraries wishing to support this functionality should call ExgNotifyGoto immediately after calling ExgNotifyReceive.

**See Also** [ExgNotifyPreview](#)

## ExgPut

**Purpose** Initiates the transfer of data to the destination device.

**Declared In** ExgMgr.h

**Prototype** Err ExgPut (ExgSocketType \*socketP)

**Parameters** -> socketP Pointer to the socket structure (see [ExgSocketType](#)). Specify either a value for the libraryRef field or a URL in the name field. libraryRef should be 0 if the name field contains a URL. The structure should also contain a value for the target, type, or name field.

**Result** Returns one of the following error codes:

errNone Success

exgErrBadLibrary Couldn't find default exchange library

## Exchange Manager

### Exchange Manager Functions

---

exgMemError	Not enough memory to initialize transfer
exgErrNotEnoughPower	The battery does not have enough power to perform the operation

Other error codes depend on the exchange library.

- Comments** Applications call this function to start a send operation. If the connection does not already exist, this function establishes one. You must create and initialize an ExgSocketType structure containing information about the data to send and the destination application. All unused fields in the structure **must** be set to 0. If no error is returned, this call **must** be followed by [ExgSend](#), to begin sending data, or [ExgDisconnect](#), to disconnect. You may need to call ExgSend multiple times to send all the data. The socket's libraryRef field or the name field must identify the library that performs the transfer. The libraryRef field identifies the exchange library by its library reference number. The name field identifies the library by URL. The socket should also specify what data is being sent by providing values for at least one of the target, name, and type fields. Use of the name field is the most common method. To provide users with a choice of transport mechanisms, the application can provide a URL that begins with a question mark (?). The Exchange Manager displays a dialog with a list of all exchange libraries that respond to URLs of the specified type. If only one exchange library is registered for this URL scheme, no dialog is displayed. For example, many applications on Palm OS 4.0 or higher support a Send command. This command generates a URL with the prefix exgSendPrefix (see [Predefined URL Prefixes](#)). The Exchange Manager displays a dialog containing a list of libraries registered for that URL scheme. The user selects an exchange library, and that library's ExgLibSend function is called. If the library is not specified by either URL or library reference number, the Exchange Manager by default uses the IR Library;

however, if the localMode flag is set, the Local Exchange Library is used instead.

<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present. Support for identifying exchange libraries by URL is implemented only if <a href="#">4.0 New Feature Set</a> is present.
----------------------	---

**See Also** [ExgDisconnect](#), [ExgSend](#), [ExgConnect](#)

## ExgReceive

<b>Purpose</b>	Receives data from a remote device.	
<b>Declared In</b>	<code>ExgMgr.h</code>	
<b>Prototype</b>	<code>UIInt32 ExgReceive (ExgSocketType *socketP, void *bufP, UInt32 bufLen, Err *err)</code>	
<b>Parameters</b>	<code>-&gt; socketP</code>	A pointer to the socket structure (see <a href="#">ExgSocketType</a> ).
	<code>&lt;- bufP</code>	A pointer to the buffer in which to receive the data.
	<code>-&gt; bufLen</code>	The number of bytes to receive.
	<code>&lt;- err</code>	A pointer to an error code result.
<b>Result</b>	<p>Returns the number of bytes actually received. A zero result indicates the end of the transmission.</p> <p>An error code is returned in the address indicated by <code>err</code>. The error code <code>exgErrUserCancel</code> is returned if the user cancels the operation. The error code <code>exgErrNotSupported</code> is returned if the application calls this function during a preview and the exchange library does not have any more data available or does not support preview.</p> <p>May display a fatal error message if the library reference number is not provided in <code>socketP</code>.</p>	

**Comments** Applications call this function in the following circumstances:

- In response to the [sysAppLaunchCmdExgReceiveData](#) launch code, following a successful call to [ExgAccept](#).
- In response to the [sysAppLaunchCmdExgPreview](#) launch code, following a successful call to ExgAccept.
- To receive requested data following a successful call to [ExgGet](#).

After receiving the data, applications call [ExgDisconnect](#) to terminate the connection.

This function blocks the application until the end of the transmission or until the requested number of bytes has been received. However, exchange libraries can provide their own user interface that is shown during this call, is updated as necessary, and allows the user to cancel the operation in progress.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present. Preview mode and [ExgGet](#) are only supported if [4.0 New Feature Set](#) is present.

**See Also** [ExgNotifyReceive](#)

## ExgRegisterDatatype

**Purpose** Registers an application to receive a specific type of data, or registers an exchange library to handle specific URL schemes.

**Declared In** ExgMgr.h

**Prototype** Err ExgRegisterDatatype (UInt32 creatorID,  
UInt16 id, const Char \*dataTypesP,  
const Char \*descriptionsP, UInt16 flags)

**Parameters**

-> creatorID	The creator ID of the registering application or exchange library.
-> id	A registry ID constant identifying the type of the items being registered. See <a href="#">Registry ID Constants</a> .

-> dataTypesP

Pointer to a tab-delimited, null-terminated string listing the items to register. (Use "\t" for the tab character.) To unregister, pass a NULL value. Each item in the string must be no more than `exgMaxTypeLength` characters. There can be no more than 16 types total.

---

**NOTE:** If specifying file extensions, do not include the period (.) that precedes the extension. If specifying URL prefixes, do not include the colon (:) at the end of the prefix.

---

-> descriptionsP

Pointer to a tab-delimited, null-terminated string that lists descriptions for the items in the `dataTypesP` parameter. (Use "\t" for the tab character.) Each description must be no longer than `exgMaxDescriptionLength`. Pass NULL to leave out the descriptions.

There must either be one description for all types or the number of descriptions must match the number of types.

The descriptions are used in dialogs displayed by Exchange Manager to identify applications or libraries.

## Exchange Manager

### Exchange Manager Functions

---

-> flags	A bit field specifying registration options. Currently, only one bit is used: the unwrap bit. Pass the exgUnwrap constant to specify that the application is registering to receive objects of this type directly if the object is sent as part of another object. For example, if a vStock object is sent as an attachment to an email message, the Exchange Manager should send the vStock object to this application directly rather than sending the message to the email application.
<b>Result</b>	Returns errNone if successful, exgMemError if there is not enough memory to save the registration info, or one of the data manager error codes (dmErr...).
<b>Comments</b>	<p>Both applications and exchange libraries use this function to register with the Exchange Manager to receive certain types of data. Applications must register with the Exchange Manager to receive data objects that do not specifically target that application using the creator ID in the target field.</p> <p>Exchange libraries register to receive data with certain URL schemes. If an exchange library is not registered to receive URLs, it only handles the receipt and sending of data if its library reference number is explicitly specified in the <a href="#">ExgSocketType</a> structure. Otherwise, the IR Library handles all incoming data for which a library could not be found.</p> <p>Both applications and libraries should register to receive data as soon as possible after they are installed and as soon as possible after a hard reset. For example, applications can call ExgRegisterDatatype in response to the <a href="#">sysAppLaunchCmdSyncNotify</a> launch code, which they receive immediately after install. Exchange libraries implemented as applications can also use this strategy. Exchange libraries implemented as shared libraries should call ExgRegisterDatatype in their startup functions.</p> <p>Make only one call to ExgRegisterDatatype per registry type. If you want to register to receive multiple items, use a tab character</p>

(\t) to separate the items. If you were to, for example, make one call to register for the DOC file extension and one call to register for the TXT extension, the second call overwrites the first. However, if you want to register with the exgUnwrap flag set, make one call without the exgUnwrap flag and one call with the exgUnwrap flag set. The application registered with the exgUnwrap flag set is stored in a different part of the registry.

Specify exgRegExtensionID to register to receive data that has a filename with a particular extension. For example, if your application wants to receive files with a TXT extension, it could register like this:

```
ExgRegisterDatatype (myCreator,  
    exgRegExtensionID, "TXT", NULL, 0);
```

If the application wants to receive files with a TXT extension or with a DOC extension, it could register like this:

```
ExgRegisterDatatype (myCreator,  
    exgRegExtensionID, "TXT\tDOC", NULL, 0);
```

Specify exgRegTypeID to register to receive data with a specific MIME type. For example, if your application wants to receive "setext" text files, it could register like this:

```
ExgRegisterDatatype (myCreator, exgRegTypeID,  
    "text/x-setext", NULL, 0);
```

Specify exgRegCreatorID to register to receive data targeted for a particular creator ID. For example, if your application wants to handle all data intended for the ToDo application, it could register like this:

```
Char ToDoCreatorStr[5];  
MemMove(ToDoCreatorStr, sysFileCToDo, 4);  
ToDoCreatorStr[4] = chrNull;  
ExgRegisterDatatype (myCreator, exgRegCreatorID,  
    ToDoCreatorStr, NULL, 0);
```

---

**NOTE:** To override one application's receipt of data, you need to also set your application as the default for this creator ID. See [ExgSetDefaultApplication](#).

---

## **Exchange Manager**

### *Exchange Manager Functions*

---

Most exchange libraries will want to register for a unique URL scheme that identifies only that library, plus they should register for a more general scheme, such as the send scheme (`exgSendScheme`), which causes the library to be listed in the Send With dialog when the user performs the Send command. The registry ID constant for URL prefixes is `exgRegSchemeID`.

```
ExgRegisterDatatype (myLibCreator,  
    exgRegSchemeID, myScheme "\t" exgSendScheme,  
    NULL, 0);
```

Registrations are active until a hard reset or until the application or library is removed. The registration information is preserved across a soft reset. When an application is removed, its registry information is also automatically removed from the registry, so there is not normally a need to unregister. If you want to unregister, you can call `ExgRegisterDatatype` with a `NULL` value for the `dataTypesP` parameter.

Multiple applications can be registered to receive the same type of data. If this is the case, the application that is registered as the default (using [ExgSetDefaultApplication](#)) is the one that receives the data unless the exchange socket explicitly specifies another application should receive it. If there is no default specified, the Exchange Manager determines a default.

Multiple libraries may also be registered to receive the same type of URL. In this case, if the URL begins with a question mark (?), the Exchange Manager displays a dialog so that the user can select which exchange library to use. If the URL does not begin with a question mark, the exchange library registered as the default is used. If there is no default specified, the Exchange Manager determines a default.

<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present. <code>ExgRegisterDatatype</code> replaces the <a href="#">ExgRegisterData</a> function.
----------------------	--

<b>See Also</b>	<a href="#">ExgRegisterData</a> , <a href="#">ExgGetTargetApplication</a> , <a href="#">ExgPut</a> , <a href="#">ExgGetDefaultApplication</a> , <a href="#">ExgGetRegisteredApplications</a> , <a href="#">ExgGetRegisteredTypes</a>
-----------------	--

## ExgRegisterData

<b>Purpose</b>	Registers an application to receive a specific type of data. This function is deprecated and replaced with <a href="#">ExgRegisterDatatype</a> .
<b>Declared In</b>	ExgMgr.h
<b>Prototype</b>	Err ExgRegisterData (UInt32 creatorID, UInt16 id, const Char *dataTypesP)
<b>Parameters</b>	-> creatorID      Creator ID of the registering application. -> id              Registry ID identifying the type of the items being registered. Specify exgRegExtensionID or exgRegTypeID. -> dataTypesP      Pointer to a tab-delimited, null-terminated string listing the items to register. (Use \t for the tab character.) These include file extensions or MIME types. To unregister, pass a NULL value.
<b>Result</b>	Returns errNone if successful, otherwise, one of the data manager error codes (dmErr...).
<b>Comments</b>	Applications that wish to receive data from anything other than another Palm Powered™ handheld running the same application must use this function to register for the kinds of data they can receive. Call this function when your application is loaded on the device.
<b>Compatibility</b>	This function corresponds to the Palm OS 3.5 version of <a href="#">ExgRegisterDatatype</a> . It is implemented only if <a href="#">3.0 New Feature Set</a> is present.

## ExgRequest

<b>Purpose</b>	Requests some data from an exchange library or an application using a URL.								
<b>Declared In</b>	ExgMgr.h								
<b>Prototype</b>	Err ExgRequest (ExgSocketType *socketP)								
<b>Parameters</b>	-> socketP Pointer to the socket structure (see <a href="#">ExgSocketType</a> ). Specify a URL in the name field and a libraryRef of 0.								
<b>Result</b>	Returns one of the following error codes:  <table><tr><td>errNone</td><td>Success</td></tr><tr><td>exgErrBadLibrary</td><td>Couldn't find default exchange library</td></tr><tr><td>exgErrNotEnoughPower</td><td>The device does not have enough power to perform the operation</td></tr><tr><td>sysErrLibNotFound</td><td>Couldn't find library or application to respond to URL</td></tr></table> Other error codes depend on the exchange library or application.	errNone	Success	exgErrBadLibrary	Couldn't find default exchange library	exgErrNotEnoughPower	The device does not have enough power to perform the operation	sysErrLibNotFound	Couldn't find library or application to respond to URL
errNone	Success								
exgErrBadLibrary	Couldn't find default exchange library								
exgErrNotEnoughPower	The device does not have enough power to perform the operation								
sysErrLibNotFound	Couldn't find library or application to respond to URL								
<b>Comments</b>	The ExgRequest function is similar to <a href="#">ExgGet</a> in that both are used to request data. The difference is that the application that calls ExgGet is always the application that receives the data. When you call ExgRequest, the application that receives the data is the application that is registered to receive it. For example, using ExgRequest, it is possible for one application to use the Exchange Manager to retrieve a vCard using any supported transport mechanism and have that data sent directly to the Address Book application instead of to the calling application.  The socketP passed to this function identifies the exchange library using a URL in the name field. The application must know beforehand the proper URL prefix for the exchange library with which it								

wants to connect. See [Predefined URL Prefixes](#) for a list of URL prefixes that the Exchange Manager provides.

If the provided URL begins with a question mark (?) and there are several exchange libraries registered for the specified URL scheme, the Exchange Manager displays a dialog from which the user selects the appropriate transport mechanism.

If the Exchange Manager cannot find a library that is registered for the specified URL, it assumes that an application is registered to receive the URL, and it launches that application with the [sysAppLaunchCmdGoToURL](#) launch code.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [ExgGet](#), [ExgNotifyReceive](#)

## ExgSend

**Purpose** Sends data to the destination device.

**Declared In** ExgMgr.h

**Prototype** `UIInt32 ExgSend (ExgSocketType *socketP,  
const void *bufP, UInt32 bufLen, Err * err)`

**Parameters**

-> socketP	A pointer to the socket structure (see <a href="#">ExgSocketType</a> ). A value must be provided for the libraryRef field. The structure should also contain values for the target, type, or name fields.
-> bufP	A pointer to the data to send.
-> bufLen	The number of bytes to send.
<- err	A pointer to an error code result.

**Result** Returns the number of bytes actually sent, normally the same number as specified in bufLen. An error code is returned in the address indicated by err. The error code exgErrUserCancel is returned if the user cancels the operation.

May display a fatal error message if the socketP parameter does not contain a value for the libraryRef field.

<b>Comments</b>	Call this function one or more times to send all the data, following a successful call to <a href="#">ExgPut</a> . After sending the data, call <a href="#">ExgDisconnect</a> to terminate the connection.  The exchange library may break large amounts of data into multiple packets or assemble small send commands together into larger packets, but the application will not be aware of these transport level details.  This function blocks the application until all the data is sent. However, the exchange library may provide its own user interface that is updated as necessary and allows the user to cancel the operation in progress.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">ExgReceive</a> , <a href="#">ExgGet</a>

## ExgSetDefaultApplication

<b>Purpose</b>	Sets the application that receives a specified type of data by default. This function also sets the default exchange library that handles particular URL schemes.						
<b>Declared In</b>	<code>ExgMgr.h</code>						
<b>Prototype</b>	<code>Err ExgSetDefaultApplication (UInt32 creatorID,                           UInt16 id, const Char *dataTypeP)</code>						
<b>Parameters</b>	<table><tr><td>-&gt; <code>creatorID</code></td><td>The creator ID of the application or library that should become the default for this type of data.</td></tr><tr><td>-&gt; <code>id</code></td><td>A registry ID constant identifying the type of data in <code>dataTypeP</code>. See <a href="#">Registry ID Constants</a>.</td></tr><tr><td>-&gt; <code>dataTypeP</code></td><td>A pointer to a null-terminated string containing the desired type of data.</td></tr></table>	-> <code>creatorID</code>	The creator ID of the application or library that should become the default for this type of data.	-> <code>id</code>	A registry ID constant identifying the type of data in <code>dataTypeP</code> . See <a href="#">Registry ID Constants</a> .	-> <code>dataTypeP</code>	A pointer to a null-terminated string containing the desired type of data.
-> <code>creatorID</code>	The creator ID of the application or library that should become the default for this type of data.						
-> <code>id</code>	A registry ID constant identifying the type of data in <code>dataTypeP</code> . See <a href="#">Registry ID Constants</a> .						
-> <code>dataTypeP</code>	A pointer to a null-terminated string containing the desired type of data.						

**NOTE:** If specifying a file extension, do not include the period (.) that precedes the extension. If specifying a URL prefix, do not include the colon (:) at the end of the prefix.

---

**Result** Returns errNone upon success or exgErrNoKnownTarget if the specified application is not registered to receive the specified data type.

**Comments** This function sets the default application that receives data of a certain type when no target is specified and the default exchange library that handles URLs with a certain prefix if no library reference number is specified.

Palm™ strongly recommends that applications allow the user to determine which application should become the default recipient for a data type. To do so, an application can use

[ExgGetRegisteredApplications](#) to get the list of applications registered for the same type of data as it is, and then display a dialog listing those applications and allow the user to select it. Then it should call ExgSetDefaultApplication with the user-specified default.

If you call ExgSetDefaultApplication with an application or library that is already the default, this function has no effect.

An application can become the default for its own creator ID even if it has not specifically registered to receive its own creator ID. That is, suppose several applications are registered to receive objects targeted for the ToDo application's creator ID. The ToDo application itself is not registered for its own creator ID, as it is not necessary to do so. However, an application can use code like the following to set the ToDo application as the default for its own creator ID.

```
Char ToDoCreatorStr[5];
MemMove(ToDoCreatorStr, sysFileCToDo, 4);
ToDoCreatorStr[4] = chrNull;
ExgSetDefaultApplication(sysFileCToDo,
    exgRegCreatorID, ToDoCreatorStr);
```

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [ExgGetDefaultApplication](#), [ExgRegisterDatatype](#)

## Application-Defined Functions

### ExgDBDeleteProcPtr

**Purpose** Handles the case where a database with an identical name already exists on the device.

**Declared In** ExgMgr.h

**Prototype** Boolean (\*ExgDBDeleteProcPtr) (const char\* nameP,  
                  UInt16 version, UInt16 cardNo, LocalID dbID,  
                  void\* userDataP)

**Parameters**

-> nameP	A pointer to the name of the identical database.
-> version	The version of the identical database.
-> cardNo	The card number of the identical database.
-> dbID	The database ID of the identical database.
-> userDataP	The userDataP parameter you passed to <a href="#">ExgDBRead</a> . If used, this parameter contains any application-specific data you find necessary. If the <a href="#">ExgDBReadProcPtr</a> function is implemented using Exchange Manager calls, this often contains the <a href="#">ExgSocketType</a> structure.

**Result** Return `true` to have the `ExgDBRead` function continue to read the database. Use this return value if you have deleted or moved the existing database or if you want the database to be overwritten. Return `false` to have `ExgDBRead` exit without reading the database.

**Comments** This function is called if the Data Manager can't create the incoming database because a database with the same name already exists. You should delete the existing database or take some other action, such as changing the database name. It is appropriate to prompt the user before choosing to delete or move the database.

## ExgDBReadProcPtr

**Purpose** Reads in the database and pass it to [ExgDBRead](#).

**Declared In** ExgMgr.h

**Prototype** Err (\*ExgDBReadProcPtr) (void\* dataP,  
UIInt32\* sizeP, void\* userDataP)

**Parameters** <- dataP A pointer to a buffer where this function should place the database data. This buffer is allocated in the dynamic heap by ExgDBRead; you don't need to use [DmWrite](#) when filling it.

<-> sizeP The size of dataP. This value is set by ExgDBRead to the number of bytes it expects to receive in dataP. You must set this value to the number of bytes you return in dataP (if it's not the same).

-> userDataP The userDataP parameter you passed to ExgDBRead. Pass the [ExgSocketType](#) structure if you implement this function using Exchange Manager calls.

**Result** Return an error number, or errNone if there is no error. If this function returns an error, ExgDBRead deletes the database it was creating, cleans up any memory it allocated, then exits, returning the error passed back from this function.

**Comments** ExgDBRead is commonly used to receive a database from a beam or from some other transport mechanism. In this case, an appropriate implementation of this callback function is to call [ExgReceive](#) as shown here:

```
Err MyReadDBProc (void *dataP, UInt32 *sizeP,
                  void *userDataP)
{
    Err err = errNone;
    //userDataP contains ExgSocketType pointer.
    *sizeP =
        ExgReceive((ExgSocketType *)userDataP,
                   dataP, *sizeP, &err);
    return err;
}
```

## ExgDBWriteProcPtr

**Purpose** Writes out the database.

**Declared In** ExgMgr.h

**Prototype** Err (\*ExgDBWriteProcPtr) (const void\* dataP,  
UInt32\* sizeP, void\* userDataP)

<b>Parameters</b>	-> dataP	A pointer to a buffer containing the database data, placed there by <a href="#">ExgDBWrite</a> .
	<-> sizeP	The number of bytes placed in dataP by ExgDBWrite. If you were unable to write out or send all of the data in this chunk, on exit, set sizeP to the number of bytes you did write.
	-> userDataP	The userDataP parameter you passed to ExgDBWrite. You can use it for application-specific data. Pass the <a href="#">ExgSocketType</a> structure if you implement this function using Exchange Manager calls.

**Result** Return an error number, or errNone if there is no error. If this function returns an error, ExgDBWrite closes the database it was reading, cleans up any memory it allocated, then exits, returning the error passed back from this function.

**Comments** ExgDBWrite is commonly used to write a database that is going to be beamed to another device (or sent through some other transport mechanism). In this case, an appropriate implementation of this callback function is to call [ExgSend](#) as shown here:

```
Err MyWriteDBProc (void *dataP, UInt32 *sizeP,
                    void *userDataP)
{
    Err err = errNone;

    //userDataP contains ExgSocketType pointer.
    *sizeP =
        ExgSend((ExgSocketType *)userDataP,
                dataP, *sizeP, &err);
    return err;
}
```

## **Exchange Manager**

*Application-Defined Functions*

---

# Exchange Library

---

The Exchange Library API described in this chapter and declared in ExgLib.h specifies a minimal set of functions that all exchange libraries must implement. This chapter is directed towards developers who use or create exchange libraries. Developers creating an exchange library should also read the [Exchange Libraries](#) chapter of the *Palm OS Programmer's Companion*, vol. II, *Communications*.

## Exchange Library Functions

### ExgLibAccept

**Purpose** Accept an incoming connection.

**Declared In** ExgLib.h

**Prototype** Err ExgLibAccept (UInt16 libRefnum,  
ExgSocketType \*exgSocketP)

**Parameters** -> libRefnum Reference number of this exchange library.  
-> exgSocketP A pointer to the socket structure (see  
[ExgSocketType](#)).

**Result** Returns errNone if no error. exgErrNotSupported is returned if a preview is in progress and the exchange library does not support preview. Other error codes are defined by each exchange library.

**Comments** The Exchange Manager's [ExgAccept](#) function simply calls ExgLibAccept in the exchange library identified by the [ExgSocketType](#) structure passed to ExgAccept. An application calls the Exchange Manager's ExgAccept function when:

## Exchange Library

### Exchange Library Functions

---

- The application wants to initiate a connection to receive data, which it does in response to `sysAppLaunchCmdExgReceiveData`.
- The application wants to initiate a connection to receive a preview of the data, which it does in response to `sysAppLaunchCmdExgAskUser`.

Any implementation of `ExgLibAccept` should update any progress dialogs to indicate that data is being accepted (or received) into an application.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [ExgLibPut](#)

## ExgLibClose

**Purpose** Library-specific. Although this function is not called by the Exchange Manager, all shared libraries normally implement it.

**Declared In** `ExgLib.h`

**Prototype** `Err ExgLibClose (UInt16 libRefnum)`

**Parameters** `-> libRefnum` Reference number of this exchange library.

**Comments** Exchange libraries are free to implement this function for internal or external use. The Exchange Manager does not call it.

## ExgLibConnect

<b>Purpose</b>	Open a connection in preparation for sending or receiving objects.				
<b>Declared In</b>	ExgLib.h				
<b>Prototype</b>	<pre>Err ExgLibConnect (UInt16 libRefNum, ExgSocketType *exgSocketP)</pre>				
<b>Parameters</b>	<table><tr><td>-&gt; libRefNum</td><td>Reference number of this exchange library.</td></tr><tr><td>&lt;-&gt; exgSocketP</td><td>A pointer to an <a href="#">ExgSocketType</a> structure identifying the socket through which objects will be sent or received.</td></tr></table>	-> libRefNum	Reference number of this exchange library.	<-> exgSocketP	A pointer to an <a href="#">ExgSocketType</a> structure identifying the socket through which objects will be sent or received.
-> libRefNum	Reference number of this exchange library.				
<-> exgSocketP	A pointer to an <a href="#">ExgSocketType</a> structure identifying the socket through which objects will be sent or received.				
<b>Result</b>	Returns errNone if no error. If ExgLibConnect is not supported by this library, this function returns exgErrNotSupported; if its use is optional, errNone is returned. Other error codes are defined by each exchange library.				
<b>Comments</b>	<p>The Exchange Manager may call this function to initiate a connection for sending multiple objects or for performing two-way communications. Some exchange libraries support sending multiple objects but do not support this call. See “<a href="#">Sending Multiple Objects</a>” on page 17 of <i>Palm OS Programmer’s Companion</i>, vol. II, <i>Communications</i> for more information.</p> <p>Not all exchange libraries support this operation. In this case, the first call to <a href="#">ExgLibPut</a> must clean up after itself before returning an error. The exchange library should not expect to get an <a href="#">ExgLibDisconnect</a> call.</p> <p>If ExgLibConnect is supported and an application calls <a href="#">ExgConnect</a>, the exchange library should delay any cleanup until ExgLibDisconnect, unless ExgLibConnect returns an error, in which case it should clean up after itself. The library can expect to get an ExgLibDisconnect call if it returns errNone from ExgLibConnect. If the application does not call ExgConnect, the first call to ExgLibPut must clean up after itself before returning an error and the exchange library should not expect to get an ExgLibDisconnect call.</p>				

## Exchange Library

### Exchange Library Functions

---

The Exchange Manager's ExgConnect function calls ExgLibConnect in the exchange library identified by the [ExgSocketType](#) structure passed to ExgConnect. If ExgLibConnect is implemented to return exgErrNotEnoughPower, the Exchange Manager puts up an alert, so there is no need for the exchange library to do so. Other error codes are not treated specially by the Exchange Manager; the exchange library must put up its own alerts when appropriate.

The exchange library may prompt the user for addressing information.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## ExgLibControl

**Purpose** Supply information about the exchange library.

**Declared In** ExgLib.h

**Prototype** Err ExgLibControl (UInt16 libRefNum, UInt16 op,  
void \*valueP, UInt16 \*valueLenP)

**Parameters** -> libRefNum Reference number of this exchange library.  
-> op The operation to perform.  
<-> valueP An operation specific parameter. See comments below.  
<-> valueLenP An operation specific parameter. See comments below.

**Result** Returns errNone if no error. exgErrNotSupported is returned if the specified operation is not supported by the exchange library. exgErrBadParam is returned if the parameters (valueP and valueLenP) are not appropriate for the operation. Additional error codes are defined by each exchange library.

**Comments** ExgLibControl is a general purpose function that performs various minor operations based upon a selector. ExgMgr.h defines

three selectors: `exgLibCtlGetTitle`, which should be supported by all exchange libraries, `exgLibCtlGetVersion`, and `exgLibCtlGetPreview`. Additional library-specific selectors should be numbered starting at `exgLibCtlSpecificOp` (0x8000).

Upon receiving `exgLibCtlGetTitle`, the exchange library must return a library title suitable for use in Exchange Manager dialogs. Exchange libraries that are built as applications should generally return their '`tAIN`' resource as their title. Be sure that the title returned through `valueP` honors the maximum length specified in `valueLenP`.

An exchange library that implements the Palm OS 4.0 version of the Exchange Library API needn't do anything special upon receiving `exgLibCtlGetVersion`; it should simply return `exgErrNotSupported`. Otherwise, set `valueP` to a two-byte value indicating the API version number and return `errNone`. Note that a version number of zero corresponds to the Palm OS 4.0 version of the Exchange Library API.

The `exgLibCtlGetPreview` operation is used by the Exchange Manager to determine whether a given exchange library supports preview. The Exchange Manager assumes that an exchange library supports preview if the exchange library doesn't implement this operation. To indicate that a library does not support preview, set `valueP` to `false` and return `errNone`.

The Exchange Manager's [ExgControl](#) function simply calls `ExgLibControl` in the exchange library identified by the [ExgSocketType](#) structure passed to `ExgControl`.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## Exchange Library

### Exchange Library Functions

---

## ExgLibDisconnect

<b>Purpose</b>	Disconnect a connection made with <a href="#">ExgLibConnect</a> , <a href="#">ExgLibAccept</a> , <a href="#">ExgLibPut</a> , or <a href="#">ExgLibGet</a> .
<b>Declared In</b>	ExgLib.h
<b>Prototype</b>	Err ExgLibDisconnect (UInt16 libRefnum, ExgSocketType *exgSocketP, Err error)
<b>Parameters</b>	-> libRefnum      Reference number of this exchange library. -> exgSocketP      A pointer to an <a href="#">ExgSocketType</a> structure identifying the socket connection to be disconnected. -> error      The current error state. Used to indicate why the connection is being broken, for example, user cancel or out of memory.
<b>Result</b>	Typically the same error code passed in. However, this function may return an error even if errNone is passed in.
<b>Comments</b>	ExgLibDisconnect may be used to finish reading the data during a preview; in this case, the connection, if any, is not shut down. Applications call ExgLibDisconnect when all data has been sent or the application wants to stop the send process. If the send data process is not completed, the caller should pass an error parameter indicating why the operation was stopped. ExgLibDisconnect is responsible for completing the operation and closing any communication ports if necessary. If data was buffered for sending in ExgLibSend, then the disconnect process may actually perform the entire transmit operation. It is important to note that the ExgLibDisconnect function can be called for an ExgLibPut, ExgLibAccept or ExgLibGet function. So it is equally important to keep track of the current operation in the <a href="#">ExgSocketType</a> . If dialogs are displayed, this function must update them as appropriate. If there are errors, ExgLibDisconnect should display them (if allowed by the application).

If [ExgLibConnect](#) is not supported, the first call to [ExgLibPut](#) must clean up after itself before returning an error and the exchange library should not expect to get an [ExgLibDisconnect](#) call.

If [ExgLibConnect](#) is supported and an application calls [ExgConnect](#), the exchange library should delay any cleanup until [ExgLibDisconnect](#), unless [ExgLibConnect](#) returns an error, in which case it should clean up after itself. The library can expect to get an [ExgLibDisconnect](#) call if it returns [errNone](#) from [ExgLibConnect](#). If the application does not call [ExgConnect](#), the first call to [ExgLibPut](#) must clean up after itself before returning an error and the exchange library should not expect to get an [ExgLibDisconnect](#) call.

The Exchange Manager's [ExgDisconnect](#) function simply calls [ExgLibDisconnect](#) in the exchange library identified by the [ExgSocketType](#) structure passed to [ExgDisconnect](#).

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## ExgLibGet

**Purpose** Establish a connection and request data from a remote device.

**Declared In** ExgLib.h

**Prototype** Err ExgLibGet(UInt16 libRefNum,  
ExgSocketType \*exgSocketP)

**Parameters** -> libRefNum Reference number of this exchange library.  
<-> exgSocketP A pointer to the socket structure (see [ExgSocketType](#)).

**Result** Returns [errNone](#) if no error. [exgErrNotSupported](#) is returned if this operation is not supported by this library. Other error codes are defined by each exchange library.

**Comments** [ExgLibGet](#) informs the library that it should make a connection to the remote device and request information from it. When an

## Exchange Library

### Exchange Library Functions

---

exchange library's `ExgLibGet` function is called, it should fetch the requested data and prepare to deliver it when the application calls `ExgReceive`. After `ExgLibReceive` is called, possibly more than once, `ExgLibDisconnect` follows.

The Exchange Manager's `ExgGet` function calls `ExgLibGet` in the exchange library identified in the `ExgSocketType` structure passed to `ExgGet`. If an exchange library's implementation of `ExgLibGet` returns `exgErrNotEnoughPower`, the Exchange Manager puts up an alert, so there is no need for the exchange library to do so. Other error codes are not treated specially by the Exchange Manager; an exchange library must put up its own alerts when appropriate.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## ExgLibHandleEvent

**Purpose** Handle exchange-library-specific events.

**Declared In** `ExgLib.h`

**Prototype** `Boolean ExgLibHandleEvent (UInt16 libRefnum,  
void *eventP)`

**Parameters** `-> libRefnum` Reference number of this exchange library.  
`-> eventP` The event to handle.

**Result** Returns true if the event was handled.

**Comments** Exchange libraries are free to implement this function for internal or external use. The Exchange Manager does not call it.

## ExgLibOpen

**Purpose** Library-specific. Although this function is not called by the Exchange Manager, all shared libraries normally implement it.

**Declared In** ExgLib.h

**Prototype** Err ExgLibOpen (UInt16 libRefnum)

**Parameters** -> libRefnum      Reference number of this exchange library.

**Comments** Exchange libraries are free to implement this function for internal or external use. The ExgLibDisconnect does not call it.

## Exchange Library

### Exchange Library Functions

---

## ExgLibPut

<b>Purpose</b>	Signals the start of an object to be transferred to the destination device.				
<b>Declared In</b>	ExgLib.h				
<b>Prototype</b>	<pre>Err ExgLibPut(UInt16 libRefnum, ExgSocketType *exgSocketP)</pre>				
<b>Parameters</b>	<table><tr><td>-&gt; libRefnum</td><td>Reference number of this exchange library.</td></tr><tr><td>&lt;- exgSocketP</td><td>Pointer to the socket structure (see <a href="#">ExgSocketType</a>).</td></tr></table>	-> libRefnum	Reference number of this exchange library.	<- exgSocketP	Pointer to the socket structure (see <a href="#">ExgSocketType</a> ).
-> libRefnum	Reference number of this exchange library.				
<- exgSocketP	Pointer to the socket structure (see <a href="#">ExgSocketType</a> ).				
<b>Result</b>	Returns errNone if no error. Error codes are defined by each exchange library.				
<b>Comments</b>	<p>Opens a connection if necessary. The actual data should be sent using <a href="#">ExgLibSend</a> after which the connection should be shut down with <a href="#">ExgLibDisconnect</a>. The exchange library may prompt the user for addressing information.</p> <p>The first time this library is called, it may be necessary to allocate global variables and perform other initialization steps. It is usually a good idea to keep any state information about the open connection in the <code>socketRef</code> field of the <code>exgSocketP</code> structure. This data can then be passed to subsequent operations on that socket.</p> <p><code>ExgLibPut</code> should then check if the <code>socketRef</code> field of <code>exgSocketP</code> has been initialized. In some cases, an application may already have filled <code>socketRef</code> with addressing information. The use of <code>socketRef</code> is entirely up to the exchange library. If <code>socketRef</code> is empty, the exchange library needs to fill in any addressing information. In order to do this, the exchange library needs to open its own dialog asking the user for whatever addressing information would be appropriate.</p> <p>Exchange libraries are responsible for any validation of data entered in the addressing dialog and may use Address Book lookup or other system features to improve the user experience.</p>				

Once the user has completed addressing and confirmed the dialog, the exchange library should, in general, call the [Progress Manager](#) to open a progress dialog that remains open during the entire put operation. The progress dialog should not be opened if the noStatus option was passed or if the transaction is in asynchronous mode.

If the exchange library is displaying dialogs, it must also look for events and pass them to the Progress Manager.

Other operations within `ExgLibPut` depend on the exchange library. The exchange library may open communications ports and establish remote links at this time. Or it may just open a stream for buffering data until a later operation. If any errors occur in this process, the exchange library is responsible for removing any progress dialogs and returning an error. If displaying progress, the exchange library may also need to display an error using the Progress Manager before returning.

---

**NOTE:** The progress dialog is converted to an error dialog and waits until the user dismisses it. This may occur in `ExgLibPut` or later in [ExgLibDisconnect](#). It depends on whether the connection was made in `ExgLibPut` or whether it used an existing connection made by a previous call to `ExgLibConnect` or `ExgLibPut`.

---

The first call to [ExgLibPut](#) must clean up after itself before returning an error if one of the following conditions exists:

- If [ExgLibConnect](#) is not supported.
- If `ExgLibConnect` is supported and an application does not call `ExgLibConnect`.

The Exchange Manager's [ExgPut](#) function calls `ExgLibPut` in the exchange library identified by the [ExgSocketType](#) structure passed to `ExgPut`. If `ExgLibPut` returns `exgErrNotEnoughPower`, the Exchange Manager puts up an alert, so there is no need for the exchange library to do so. Other error codes are not treated specially by the Exchange Manager; an exchange library must put up its own alerts when appropriate.

## Exchange Library

### Exchange Library Functions

---

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## ExgLibReceive

**Purpose** Receive data from a remote device.

**Declared In** ExgLib.h

**Prototype** `UInt32 ExgLibReceive(UInt16 libRefNum,  
ExgSocketType *exgSocketP, void *bufP,  
UInt32 bufSize, Err *errP)`

**Parameters**

-> libRefNum	Reference number of this exchange library.
<-> exgSocketP	A pointer to the socket structure (see <a href="#">ExgSocketType</a> ).
-> bufP	A pointer to a buffer into which the data is put.
-> bufSize	The size of the buffer in bytes.
<- errP	The error code result: errNone if no error. exgErrNotSupported is returned if used during a preview and the exchange library does not support preview or if there appears to be more data available but ExgLibReceive cannot obtain it. Error codes are defined by each exchange library.

**Result** The number of bytes received. Returns 0 if the object is complete. ExgLibReceive blocks until at least one byte is available or the object is complete.

**Comments** Use after [ExgLibGet](#) or [ExgLibAccept](#) to receive the contents of the object. May be used after ExgLibAccept to examine the contents during a preview.

ExgLibReceive must update any progress dialogs to indicate that data is being received. The ExgLibReceive should fill the buffer passed as much as possible and return the actual number of bytes that were stored in the buffer. ExgLibReceive must block for at

least one byte if the stream is not complete. Returning zero bytes indicates the end of the data.

The Exchange Manager's [ExgReceive](#) function simply calls ExgLibReceive in the exchange library identified by the [ExgSocketType](#) structure passed to ExgReceive.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## ExgLibRequest

**Purpose** Requests an object using a URL, then has the Exchange Manager send it to the default application registered for the object's type.

**Declared In** ExgLib.h

**Prototype** Err ExgLibRequest (UInt16 libRefNum,  
ExgSocketType \*socketP)

**Parameters** -> libRefNum Reference number of this exchange library.  
<-> socketP Pointer to the socket structure (see [ExgSocketType](#)). The name field contains the URL of the object being requested.

**Result** Returns errNone if no error. exgErrNotSupported is returned if the exchange library does not support this operation. Other error codes are defined by each exchange library.

**Comments** The exchange library may prompt the user for addressing information. The socket's name field may contain a URL with any of the schemes for which the exchange library registered. Not all exchange libraries support this operation.

---

**NOTE:** This function is often used to tickle an exchange library to make it check to see if there are new messages. These messages are then delivered as usual. So there may or may not be a specific object being requested.

---

## Exchange Library

### Exchange Library Functions

---

The Exchange Manager's [ExgRequest](#) function calls `ExgLibRequest` in the exchange library identified by the `ExgSocketType` structure passed to `ExgRequest`. If `ExgLibRequest` returns `exgErrNotEnoughPower`, the Exchange Manager puts up an alert, so there is no need for the exchange library to do so. Other error codes are not treated specially by the Exchange Manager; an exchange library must put up its own alerts when appropriate.

**See Also** [ExgRegisterDatatype](#)

## ExgLibSend

**Purpose** Send data for an object to a destination device.

**Declared In** `ExgLib.h`

**Prototype** `UInt32 ExgLibSend(UInt16 libRefNum,  
ExgSocketType *exgSocketP, const void *bufP,  
UInt32 bufLen, Err *errP)`

**Parameters**

-> libRefNum	Reference number of this exchange library.
<- exgSocketP	A pointer to the socket structure (see <a href="#">ExgSocketType</a> ).
-> bufP	A pointer to a buffer containing the data to send.
-> bufLen	The number of bytes to send.
<- errP	The error code result. Error codes are defined by each exchange library

**Result** Returns the number of bytes sent.

**Comments** Applications call [ExgSend](#) after [ExgPut](#) and in response to `sysAppLaunchCmdExgGetData`.

This function blocks until all bytes to be sent are actually sent, or until an error occurs (such as device full).

ExgLibSend may be called any number of times with varying size buffers to transmit information. If dialogs are being displayed, ExgLibSend must keep them updated (perhaps with animation or progress information). ExgLibSend must also check for events and let the Progress Manager handle them.

The Exchange Manager's ExgSend function simply calls ExgLibSend in the exchange library identified by the [ExgSocketType](#) structure passed to ExgSend.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## ExgLibSleep

**Purpose** The device is going to sleep.

**Declared In** ExgLib.h

**Prototype** Err ExgLibSleep (UInt16 libRefnum)

**Parameters** -> libRefnum Reference number of this exchange library.

**Result** Returns errNone if no error. Error codes are defined by each exchange library.

**Comments** The device goes into sleep mode when the user turns the device off, the auto-off timer expires, or power is low. All shared libraries must implement ExgLibSleep; however, no processing is required and simply returning errNone is enough.

---

**IMPORTANT:** Libraries must return from this function quickly to allow sufficient time for emergency shutdown situations (removal of batteries, for example).

---

## **Exchange Library**

### *Exchange Library Functions*

---

## **ExgLibWake**

**Purpose** The device is waking up.

**Declared In** ExgLib.h

**Prototype** Err ExgLibWake (UInt16 libRefnum)

**Parameters** -> libRefnum Reference number of this exchange library.

**Result** Returns errNone if no error. Error codes are defined by each exchange library.

**Comments** The device wakes up when the user turns the device on. All shared libraries must implement ExgLibWake, although no processing is required; it is enough to simply return errNone.

# IR Library

---

The IR (InfraRed) library is a shared library that provides a direct interface to the IR communications capabilities of the Palm OS®. This chapter provides reference material for the IR library API:

- [IR Library Data Structures](#)
- [IR Library Constants](#)
- [IR Stack Callback Events](#)
- [IR Library Functions](#)
- [IAS Functions](#)
- [Application-Defined Functions](#)

The header file `irlib.h` declares the IR library API. For more information on the IR library, see the chapter “[Beaming \(Infrared Communication\)](#)” in the *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## IR Library Data Structures

This section lists some of the more important data types used by the IR library functions.

### **IrConnect**

The `IrConnect` structure is used to manage an IrLMP or Tiny TP connection.

```
typedef struct _hconnect {  
    UInt8 lLsap;  
    UInt8 rLsap;  
    UInt8 flags;  
    UInt8 reserved;  
    IrCallBack callBack;  
    IrPacket packet;  
    ListEntry packets;
```

## IR Library

### *IR Library Data Structures*

---

```
    UInt16 sendCredit;
    UInt8 availCredit;
    UInt8 dataOff;
} _hconnect;
```

#### **Field Descriptions**

lLsap	The local LSAP on which this connection listens.
rLsap	The LSAP assigned to the remote side.
flags	For system use only.
reserved	Reserved for future use.
callBack	Pointer to callback function. For system use only.
packet	For system use only.
packets	List of packets to send.
sendCredit	Amount of credit from peer.
availCredit	Amount of credit to give to peer.
dataOff	Amount of data less than IrLAP size.

## **IrPacket**

The `IrPacket` structure is used for sending IrDA packets.

```
typedef struct _IrPacket {
    ListEntry node;
    UInt8 *buff;
    UInt16 len;
    IrConnect* origin;
    UInt8 headerLen;
    UInt8 header[14];
    UInt8 reserved;
} IrPacket;
```

#### **Field Descriptions**

node	For system use only.
buff	Pointer to the send data buffer.

len	Number of bytes in data buffer.
origin	Pointer to connection that owns the packet. For system use only.
headerLen	Number of bytes contained in the header. For system use only.
header	Storage for header. For system use only.
reserved	Reserved for future use.

---

**IMPORTANT:** The node field must be the first field in the structure. It is used internally by the stack.

---

## IrlIASObject

The IrlIASObject structure is used as storage for an IAS object managed by the local IAS server. An object of this type is passed as the obj parameter to the [IrIAS\\_Add](#) function.

```
typedef struct _IrlIasObject {  
    UInt8 *name;  
    UInt8 len;  
    UInt8 nAttribs;  
    IrIasAttribute* attribs;  
} IrlIasObject;
```

### Field Descriptions

name	Pointer to name of object.
len	Length of object name.
nAttribs	Number of attributes.
attribs	Pointer to an array of attributes.

## IrlasQuery

The `IrlasQuery` structure is used to perform IAS queries. The `IrlasQuery` object is passed as the `token` parameter to functions such as `IrlIAS_Query` and `IrlIAS_Next`.

```
typedef struct _IrlasQuery {
    UInt8 queryLen;
    UInt8 reserved;
    UInt8 *queryBuf;
    UInt16 resultBufSize;
    UInt16 resultLen;
    UInt16 listLen;
    UInt16 offset;
    UInt8 retCode;
    UInt8 overFlow;
    UInt8 *result;
    IrlasQueryCallBack callBack;
} _IrlasQuery;
```

### Field Descriptions

queryLen	Total length of the query.
reserved	Reserved for future use.
queryBuf	Pointer to buffer containing the query.
resultBufSize	Size of the result buffer.
resultLen	Actual number of bytes in the result buffer.
listLen	Number of items in the result list.
offset	Offset into the results buffer.
retCode	Return code of operation.
overFlow	Set to true if result exceeded result buffer size.
result	Pointer to buffer containing result.
callBack	Pointer to query callback function.

## IrCallbackParms

The `IrCallbackParms` structure is used to pass information from the stack to the upper layer of the stack (application). Not all fields are valid at any given time. The type of event determines which fields are valid. The `IrCallbackParms` object is passed as the second parameter to the `IrCallback` function.

```
typedef struct {
    IrEvent event;
    UInt8 reserved1;
    UInt8 *rxBuff;
    UInt16 rxLen;
    IrPacket* packet;
    IrDeviceList* deviceList;
    IrStatus status;
    UInt8 reserved2;
} IrCallBackParms;
```

### Field Descriptions

event	Event causing the callback.
reserved1	Reserved for future use.
rxBuff	Received data buffer.
rxLen	Length of data in received buffer.
packet	Pointer to packet being returned.
deviceList	Pointer to discovery device list.
status	Status of stack.
reserved2	Reserved for future use.

## IrStatsType

The `IrStatsType` structure defines performance statistics for the IR Library. Use the [ExgControl](#) function with an `irGetStatistics` operation to retrieve these statistics. See [IR Control Constants](#) for more information.

```
typedef struct {
    UInt16 recLineErrors;
    UInt16 crcErrors;
} IrStatsType;
```

### Field Descriptions

- recLineErrors The number of serial errors since the library opened.
- crcErrors The number of CRC errors since the library opened.

## IR Library Constants

### IR Control Constants

The IR control constants define operations that the IR Exchange Library can perform. You pass these constants as the operation parameter to [ExgControl](#). The following table lists the operation constants, the data that should be passed as the valueP parameter to ExgControl, and what operation is performed in response.

Operation Constant	value Data Type	Description
irGetScanningMode	Boolean. Output only.	Returns true in *valueP if beaming is enabled or false if beaming is disabled.
irGetStatistics	<a href="#">IrStatsType</a> . Output only.	Returns performance statistics.
irRestoreScanning	None	Re-enables beaming after an irSuppressScanning operation. This operation keeps track of the number of requests that beaming be disabled and re-enables beaming only when the count reaches 0.

<b>Operation Constant</b>	<b>value Data Type</b>	<b>Description</b>
		This operation differs from <code>irSetScanningMode</code> in that it does not update the saved preferences.
<code>irSetScanningMode</code>	Boolean. Input only.	Enables or disables beaming.
		This operation modifies the saved preferences database, which is back up during a HotSync® operation. Because of this, beaming may remain disabled after a reset if you use this operation to disable it. If you want to temporarily disable beaming use <code>irSuppressScanning</code> and <code>irRestoreScanning</code> instead.
<code>irSetBaudMask</code>	UInt16 containing a mask of the <code>irOpenOptSpeed...</code> constants defined in <code>IrLib.h</code> . Input only.	Sets the possible baud rates that the IR Library will use to those specified in <code>*valueP</code> . OR the <code>irOpenOptSpeed...</code> constants together to specify more than one. The default rate is 0, which causes the baud rate to be determined by the hardware.
		This operation is sometimes useful for debugging connections. Generally, you should set all bits up to the fastest rate you want to allow. To reset, use this operation again and pass 0 in <code>*valueP</code> .
		If you change the baud rate, your changes are until the device is reset or you perform this operation again.

## IR Library

### IR Stack Callback Events

---

Operation Constant	value Data Type	Description
irSetSerialMode	Boolean. Input only.	If the specified value is <code>true</code> , the IR Library uses the serial port instead of the infrared port until the device is reset. This option is useful for debugging. You can run your application in POSE and use the IR Library to communicate with a device connected in the cradle.
irSetSupported	Boolean. Input only.	If <code>true</code> , IR is supported on this device. If <code>false</code> , IR is not supported. You can use this constant to disable the unsupported dialog that normally displays when a beam is attempted and no IR support is available.
irSuppressScanning	None	Temporarily disables beam receive. This operation keeps track of the number of requests that beaming be disabled and re-enables beaming (through <code>irRestoreScanning</code> ) only when the count reaches 0.  This operation differs from <code>irSetScanningMode</code> in that it does not update the saved preferences.

---

## IR Stack Callback Events

The IR stack calls the application by way of a callback function stored in each `IrConnect` structure. The callback function is called with a pointer to the `IrConnect` structure and a pointer to a parameter structure. The parameter structure contains an event field, which indicates the reason the callback is called, and other parameters, which have meaning based on the event.

The meaning of the events is described in the following sections.

### **LEVENT\_DATA\_IND**

Data has been received. The received data is accessed using fields rxBuff and rxLen.

### **LEVENT\_DISCOVERY\_CNF**

Indicates the completion of a discovery operation. The field deviceList points to the discovery list.

### **LEVENT\_LAP\_CON\_CNF**

The requested IrLAP connection has been made successfully. The callback function of all bound IrConnect structures is called.

### **LEVENT\_LAP\_CON\_IND**

Indicates that the IrLAP connection has come up. The callback of all bound IrConnect structures is called.

### **LEVENT\_LAP\_DISCON\_IND**

Indicates that the IrLAP connection has gone down. This means that all IrLMP connections are also down. A callback with event LEVENT\_LM\_CON\_IND is not given. The callback function of all bound IrConnect structures is called.

### **LEVENT\_LM\_CON\_CNF**

The requested IrLMP/Tiny TP connection has been made successfully. Connection data from the other side is found using fields rxBuff and rxLen.

### **LEVENT\_LM\_CON\_IND**

Other device has initiated a connection. IrConnectRsp should be called to accept the connection. Any data associated with the connection request can be found using fields rxBuff and rxLen, data pointer and length, respectively.

## **LEVENT\_LM\_DISCON\_IND**

The IrLMP/Tiny TP connection has been disconnected. Any data associated with the disconnect indication can be found using fields rxBuff and rxLen, data pointer and length, respectively.

## **LEVENT\_PACKET\_HANDLED**

A packet is being returned. A pointer to the packet exists in field packet.

## **LEVENT\_STATUS\_IND**

Indicates that a status event from the stack has occurred. The status field indicates the status generating the event. Possible status values are as follows:

- IR\_STATUS\_NO\_PROGRESS which means that IrLAP has no progress for 3 seconds threshold time (for example, the beam is blocked).
- IR\_STATUS\_LINK\_OK which indicates that the no progress condition has cleared.
- IR\_STATUS\_MEDIA\_NOT\_BUSY which indicates that the IR media has transitioned from busy to not busy.

## **LEVENT\_TEST\_CNF**

Indicates that a TEST command has completed. The status field indicates if the test was successful.

- IR\_STATUS\_SUCCESS indicates that the operation was successful and the data in the test response can be found by using the rxBuff and rxLen fields.
- IR\_STATUS\_FAILED indicates that no TEST response was received. The packet passed to perform the test command is passed back in the packet field and is now available (no separate packet handled event occurs).

## **LEVENT\_TEST\_IND**

Indicates that a TEST command frame has been received. A pointer to the received data is in rxBuff and rxLen. A pointer to the

packet that is sent in response to the test command is in the packet field. The packet is currently set up to respond with the same data sent in the command TEST frame. If different data is desired as a response, then you need to modify the packet structure. This event is sent to the callback function in all bound IrConnect structures. The IAS connections ignore this event.

## IR Library Functions

### **IrAdvanceCredit**

**Purpose** Advances the credit to the other side of the connection.

**Declared In** IrLib.h

**Prototype** void IrAdvanceCredit (IrConnect\* con,  
                  UInt8 credit)

**Parameters**     --> con                      Pointer to [IrConnect](#) structure representing  
  connection to which credit is advanced.  
           --> credit                        Amount of credit to advance.

**Result** Returns nothing.

**Comments** The credit passed by this function is added to the existing available credit, which must not exceed 127. This function only makes sense for a Tiny TP connection.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

## IrBind

<b>Purpose</b>	Obtains a local LSAP selector and registers the connection with the protocol stack.						
<b>Declared In</b>	<code>IrLib.h</code>						
<b>Prototype</b>	<code>IrStatus IrBind (UInt16 refNum, IrConnect* con, IrCallBack callBack)</code>						
<b>Parameters</b>	<table><tr><td><code>--&gt; refnum</code></td><td>IR library refNum.</td></tr><tr><td><code>&lt;-&gt; con</code></td><td>Pointer to <a href="#">IrConnect</a> structure.</td></tr><tr><td><code>--&gt; callBack</code></td><td>Pointer to a callBack function that handles the indications and confirmation from the protocol stack.</td></tr></table>	<code>--&gt; refnum</code>	IR library refNum.	<code>&lt;-&gt; con</code>	Pointer to <a href="#">IrConnect</a> structure.	<code>--&gt; callBack</code>	Pointer to a callBack function that handles the indications and confirmation from the protocol stack.
<code>--&gt; refnum</code>	IR library refNum.						
<code>&lt;-&gt; con</code>	Pointer to <a href="#">IrConnect</a> structure.						
<code>--&gt; callBack</code>	Pointer to a callBack function that handles the indications and confirmation from the protocol stack.						
<b>Result</b>	<p><code>IR_STATUS_SUCCESS</code> means the operation completed successfully. The assigned LSAP can be found in <code>con-&gt;lLsap</code>.</p> <p><code>IR_STATUS_FAILED</code> means the operation failed for one of the following reasons:</p> <ul style="list-style-type: none"><li>• <code>con</code> is already bound to the stack.</li><li>• There is no room in the connection table.</li></ul>						
<b>Comments</b>	The <code>IrConnect</code> structure is re-initialized. Any values stored in the structure are lost. The assigned LSAP is returned in the <code>lLsap</code> field of <code>con</code> . The type of the connection is set to <code>IrLMP</code> . The <code>IrConnect</code> must be bound to the stack before it can be used.						
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.						

## **IrClose**

**Purpose** Closes the IR library. This releases the global memory for the IR stack and any system resources it uses. This must be called when an application is done with the IR library.

**Declared In** IrLib.h

**Prototype** Err IrClose (UInt16 refnum)

**Parameters** --> refnum      IR library refNum.

**Result** Returns 0 if successful.

**Comments** Do not call this function unless the call to [IrOpen](#) was successful.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

## **IrConnectIrLap**

**Purpose** Starts an IrLAP connection.

**Declared In** IrLib.h

**Prototype** IrStatus IrConnectIrLap (UInt16 refNum,  
IrDeviceAddr deviceAddr)

**Parameters** --> refnum      IR library refNum.

--> deviceAddr      32-bit address of device to which connection  
should be made.

**Result** IR\_STATUS\_PENDING means the operation started successfully;  
the result is returned by way of a callback.

IR\_STATUS\_MEDIA\_BUSY means the operation failed because the  
media is busy. Media busy is caused by one of the following  
reasons:

- Other devices are using the IR medium.

## IR Library

### IR Library Functions

---

- An IrLAP connection already exists.
- A discovery process is in progress.

<b>Comments</b>	The result is signaled to all bound <code>IrConnect</code> structures by way of the callback function. The callback event is <code>LEVENT_LAP_CON_CNF</code> if successful or <code>LEVENT_LAP_DISCON_IND</code> if unsuccessful.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## IrConnectReq

<b>Purpose</b>	Requests an IrLMP or Tiny TP connection.	
<b>Declared In</b>	<code>IrLib.h</code>	
<b>Prototype</b>	<code>IrStatus IrConnectReq (UInt16 refNum, IrConnect* con, IrPacket* packet, UInt8 credit)</code>	
<b>Parameters</b>	<code>--&gt; refnum</code>	IR library refNum.
	<code>--&gt; con</code>	Pointer to <code>IrConnect</code> structure for handling the connection. The <code>rLsap</code> field must contain the LSAP selector for the peer on the other device. Also the type of the connection must be set. Use <code>IR_SetConTypeLMP</code> to set the type to an IrLMP connection or <code>IR_SetConTypeTTP</code> to set the type to a Tiny TP connection.
	<code>--&gt; packet</code>	Pointer to a packet that contains connection data. Even if no connection data is needed, the packet must point to a valid <code>IrPacket</code> structure. The packet is returned by way of the callback function with the <code>LEVENT_PACKET_HANDLED</code> event if no errors occur. The maximum size of the packet is <code>IR_MAX_CON_PACKET</code> for an IrLMP connection or <code>IR_MAX_TTP_CON_PACKET</code> for a Tiny TP connection.

--> credit

Initial amount of credit advanced to the other side. Must be less than 127. It is ANDed with 0x7f, so if it is greater than 127, unexpected results occur. This parameter is ignored if the connection is an IrLMP connection.

<b>Result</b>	<p>IR_STATUS_PENDING means the operation has been started successfully and the result is returned by way of the callback function with the event LEVENT_LM_CON_CNF if the connection is made or LEVENT_LM_DISCON_IND if connection fails. The packet is returned by way of the callback with the event LEVENT_PACKET_HANDLED.</p> <p>IR_STATUS_FAILED means the operation failed because of one of the following reasons. Note that the packet is available immediately.</p> <ul style="list-style-type: none"><li>• The connection is busy (already involved in a connection).</li><li>• The IrConnect structure is not bound to the stack.</li><li>• The packet size exceeds maximum allowed.</li></ul> <p>IR_STATUS_NO_IRLAP means the operation failed because there is no IrLAP connection (the packet is available immediately).</p>
<b>Comments</b>	The result is signaled by way of the callback specified in the IrConnect structure. The callback event LEVENT_LM_CON_CNF indicates that the connection is up and LEVENT_LM_DISCON_IND indicates that the connection failed. Before calling this function the fields in the con structure must be properly set.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## **IrConnectRsp**

<b>Purpose</b>	Accepts an incoming connection that has been signaled by way of the callback with the event <code>LEVENT_LM_CON_IND</code> .								
<b>Declared In</b>	<code>IrLib.h</code>								
<b>Prototype</b>	<code>IrStatus IrConnectRsp (UInt16 refNum, IrConnect* con, IrPacket* packet, UInt8 credit)</code>								
<b>Parameters</b>	<table><tr><td>--&gt; refnum</td><td>IR library refNum.</td></tr><tr><td>--&gt; con</td><td>Pointer to <a href="#">IrConnect</a> structure.</td></tr><tr><td>--&gt; packet</td><td>Pointer to a packet that contains connection data. Even if no connection data is needed, the packet must point to a valid <a href="#">IrPacket</a> structure. The packet is returned by way of the callback with the <code>LEVENT_PACKET_HANDLED</code> event if no errors occur. The maximum size of the packet is <code>IR_MAX_CON_PACKET</code> for an IrLMP connection or <code>IR_MAX_TTP_CON_PACKET</code> for a Tiny TP connection.</td></tr><tr><td>--&gt; credit</td><td>Initial amount of credit advanced to the other side. Must be less than 127. It is ANDed with 0x7f, so if it is greater than 127, unexpected results occur. This parameter is ignored if the connection is an IrLMP connection.</td></tr></table>	--> refnum	IR library refNum.	--> con	Pointer to <a href="#">IrConnect</a> structure.	--> packet	Pointer to a packet that contains connection data. Even if no connection data is needed, the packet must point to a valid <a href="#">IrPacket</a> structure. The packet is returned by way of the callback with the <code>LEVENT_PACKET_HANDLED</code> event if no errors occur. The maximum size of the packet is <code>IR_MAX_CON_PACKET</code> for an IrLMP connection or <code>IR_MAX_TTP_CON_PACKET</code> for a Tiny TP connection.	--> credit	Initial amount of credit advanced to the other side. Must be less than 127. It is ANDed with 0x7f, so if it is greater than 127, unexpected results occur. This parameter is ignored if the connection is an IrLMP connection.
--> refnum	IR library refNum.								
--> con	Pointer to <a href="#">IrConnect</a> structure.								
--> packet	Pointer to a packet that contains connection data. Even if no connection data is needed, the packet must point to a valid <a href="#">IrPacket</a> structure. The packet is returned by way of the callback with the <code>LEVENT_PACKET_HANDLED</code> event if no errors occur. The maximum size of the packet is <code>IR_MAX_CON_PACKET</code> for an IrLMP connection or <code>IR_MAX_TTP_CON_PACKET</code> for a Tiny TP connection.								
--> credit	Initial amount of credit advanced to the other side. Must be less than 127. It is ANDed with 0x7f, so if it is greater than 127, unexpected results occur. This parameter is ignored if the connection is an IrLMP connection.								
<b>Result</b>	<p><code>IR_STATUS_PENDING</code> means the operation has been started successfully and the packet is returned by way of the callback function with the event <code>LEVENT_PACKET_HANDLED</code>.</p> <p><code>IR_STATUS_FAILED</code> means the operation failed because of one of the following reasons. Note that the packet is available immediately.</p> <ul style="list-style-type: none"><li>• The connection is not in the proper state to require a response.</li><li>• The <code>IrConnect</code> structure is not bound to the stack.</li></ul>								

- The packet size exceeds the maximum allowed.

`IR_STATUS_NO_IRLAP` means the operation failed because there is no IrLAP connection (the packet is available immediately).

**Comments** `IrConnectRsp` can be called during the callback or later to accept the connection. The type of the connection must already have been set to IrLMP or Tiny TP before the `LEVENT_LM_CON_IND` event.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

## IrDataReq

**Purpose** Sends a data packet.

**Declared In** `IrLib.h`

**Prototype** `IrStatus IrDataReq (UInt16 refNum,  
IrConnect* con, IrPacket* packet)`

**Parameters** `--> refnum` IR library refNum.  
`--> con` Pointer to [IrConnect](#) structure that specifies the connection over which the packet should be sent.  
`--> packet` Pointer to a valid [IrPacket](#) structure that contains data to send. The packet should not exceed the maximum size found with [IrMaxTxSize](#).

**Result** `IR_STATUS_PENDING` means the packet has been queued by the stack. The packet is returned by way of the callback with event `LEVENT_PACKET_HANDLED`.

`IR_STATUS_FAILED` means the operation failed because of one of the following reasons. Note that the packet is available immediately.

- The `IrConnect` structure is not bound to the stack.
- The packet size exceeds the maximum allowed.

- The IrConnect structure does not represent an active connection.

<b>Comments</b>	The packet is owned by the stack until it is returned by way of the callback with event LEVENT_PACKET_HANDLED. The largest packet that can be sent is found by calling <a href="#">IrMaxTxSize</a> .
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## **IrDisconnectIrLap**

<b>Purpose</b>	Disconnects an IrLAP connection.
<b>Declared In</b>	<code>IrLib.h</code>
<b>Prototype</b>	<code>IrStatus IrDisconnectIrLap (UInt16 refNum)</code>
<b>Parameters</b>	<code>--&gt; refnum</code> IR library refNum.
<b>Result</b>	<code>IR_STATUS_PENDING</code> means the operation started successfully and all bound <code>IrConnect</code> structures are called back when complete. <code>IR_STATUS_NO_IRLAP</code> means the operation failed because no IrLAP connection exists.
<b>Comments</b>	When the IrLAP connection goes down, the callback of all bound <code>IrConnect</code> structures is called with event <code>LEVENT_LAP_DISCON_IND</code> .
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## IrDiscoverReq

<b>Purpose</b>	Starts an IrLMP discovery process.				
<b>Declared In</b>	<code>IrLib.h</code>				
<b>Prototype</b>	<code>IrStatus IrDiscoverReq (UInt16 refNum, IrConnect* con)</code>				
<b>Parameters</b>	<table><tr><td><code>--&gt; refnum</code></td><td>IR library refNum.</td></tr><tr><td><code>--&gt; con</code></td><td>Pointer to a bound <a href="#">IrConnect</a> structure.</td></tr></table>	<code>--&gt; refnum</code>	IR library refNum.	<code>--&gt; con</code>	Pointer to a bound <a href="#">IrConnect</a> structure.
<code>--&gt; refnum</code>	IR library refNum.				
<code>--&gt; con</code>	Pointer to a bound <a href="#">IrConnect</a> structure.				
<b>Result</b>	<p><code>IR_STATUS_PENDING</code> means the operation is started successfully; the result is returned by way of callback.</p> <p><code>IR_STATUS_MEDIA_BUSY</code> means the operation failed because the media is busy. Media busy is caused by one of the following reasons:</p> <ul style="list-style-type: none"><li>• Other devices are using the IR medium.</li><li>• A discovery process is already in progress.</li><li>• An IrLAP connection exists.</li></ul> <p><code>IR_STATUS_FAILED</code> means the operation failed because the <code>IrConnect</code> structure is not bound to the stack.</p>				
<b>Comments</b>	The result is signaled by way of the callback function specified in the <code>IrConnect</code> structure with the event <code>LEVENT_DISCOVERY_CNF</code> . Only one discovery can be invoked at a time.				
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.				

## **IrIsIrLapConnected**

<b>Purpose</b>	Determines if an IrLAP connection exists.
<b>Declared In</b>	<code>IrLib.h</code>
<b>Prototype</b>	<code>BOOL IrIsIrLapConnected (UInt16 refNum)</code>
<b>Parameters</b>	--> <code>refnum</code> IR library refNum.
<b>Result</b>	<code>true</code> if IrLAP is connected, <code>false</code> otherwise.
<b>Comments</b>	Only available if <code>IR_IS_LAP_FUNCS</code> is defined.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## **IrIsMediaBusy**

<b>Purpose</b>	Determines if the IR media is busy.
<b>Declared In</b>	<code>IrLib.h</code>
<b>Prototype</b>	<code>BOOL IrIsMediaBusy (UInt16 refNum)</code>
<b>Parameters</b>	--> <code>refnum</code> IR library refNum.
<b>Result</b>	<code>true</code> if IR media is busy, <code>false</code> otherwise.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## IrlsNoProgress

<b>Purpose</b>	Determines if IrLAP is not making progress.
<b>Declared In</b>	<code>IrLib.h</code>
<b>Prototype</b>	<code>BOOL IrIsNoProgress (UInt16 refNum)</code>
<b>Parameters</b>	<code>--&gt; refnum</code> IR library refNum.
<b>Result</b>	<code>true</code> if IrLAP is not making progress, <code>false</code> otherwise.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## IrlsRemoteBusy

<b>Purpose</b>	Determines if IrLAP of the other device is busy.
<b>Declared In</b>	<code>IrLib.h</code>
<b>Prototype</b>	<code>BOOL IrIsRemoteBusy (UInt16 refNum)</code>
<b>Parameters</b>	<code>--&gt; refnum</code> IR library refNum.
<b>Result</b>	<code>true</code> if IrLAP of the other device is busy, <code>false</code> otherwise.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## IrLocalBusy

<b>Purpose</b>	Sets the IrLAP local busy flag.
<b>Declared In</b>	<code>IrLib.h</code>
<b>Prototype</b>	<code>void IrLocalBusy (UInt16 refNum, BOOL flag)</code>
<b>Parameters</b>	<code>--&gt; refnum</code> IR library refNum.

--> flag                    Value (`true` or `false`) to set for local busy flag of IrLAP.

**Result**    Returns nothing.

**Comments**    If local busy is set to `true`, then the local IrLAP layer sends RNR (Receive Not Ready) frames to the other side indicating it cannot receive any more data. If the local busy is set to `false`, IrLAP is ready to receive frames.

The setting takes effect the next time IrLAP sends an RR (Receive Ready) frame. If IrLAP has data to send, the data is sent first, so it should be used carefully.

This function should not be used when using Tiny TP or when multiple connections exist.

**Compatibility**    Implemented only if [3.0 New Feature Set](#) is present.

## IrMaxRxSize

**Purpose**    Returns the maximum size buffer that can be sent by the other device.

**Declared In**    `IrLib.h`

**Prototype**    `UInt16 IrMaxRxSize (UInt16 refNum,  
                  IrConnect* con)`

**Parameters**    `--> refnum`            IR library refNum.  
                  `--> con`                    Pointer to [`IrConnect`](#) structure that represents an active connection.

**Result**    Returns the maximum size buffer that can be sent by the other device (maximum bytes that can be received). The value returned is only valid for active connections. The maximum size varies for each connection and is based on the negotiated IrLAP parameters and the type of the connection.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

## **IrMaxTxSize**

**Purpose** Returns the maximum size allowed for a transmit packet.

**Declared In** IrLib.h

**Prototype** `UInt16 IrMaxTxSize (UInt16 refNum,  
IrConnect* con)`

**Parameters** `--> refnum` IR library refNum.  
`--> con` Pointer to [IrConnect](#) structure that represents an active connection.

**Result** Returns the maximum size allowed for a transmit packet. The value returned is only valid for active connections. The maximum size varies for each connection and is based on the negotiated IrLAP parameters and the type of the connection.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

## **IrOpen**

**Purpose** Opens the IR library. This allocates the global memory for the IR stack and reserves the system resources it requires. This must be done before any other IR library calls are made.

**Declared In** IrLib.h

**Prototype** `Err IrOpen (UInt16 refnum, UInt32 options)`

**Parameters** `--> refnum` IR library refNum. This value is returned from the function [SysLibFind](#), which you must call first to load the IR library.

--> options      Open options flags. See the Comments section for details.

**Result**      Returns 0 if successful.

**Comments**      The following flags can be specified for the options parameter to set the speed of the connection:

irOpenOptSpeed115200      Set to maximum negotiated baud rate.  
irOpenOptSpeed57600      Set to 57600 bps (default if no flags given).  
irOpenOptSpeed9600      Set to 9600 bps.

**Compatibility**      Implemented only if [3.0 New Feature Set](#) is present.

## **IrSetConTypeLMP**

**Purpose**      Sets the type of the connection to IrLMP. This function must be called after the IrConnect structure is bound to the stack.

**Declared In**      IrLib.h

**Prototype**      void IrSetConTypeLMP (IrConnect\* con)

**Parameters**      --> con      Pointer to [IrConnect](#) structure.

**Result**      Returns nothing.

**Compatibility**      Implemented only if [3.0 New Feature Set](#) is present.

## **IrSetConTypeTTP**

<b>Purpose</b>	Sets the type of the connection to Tiny TP. This function must be called after the IrConnect structure is bound to the stack.
<b>Declared In</b>	<code>IrLib.h</code>
<b>Prototype</b>	<code>void IrSetConTypeTTP (IrConnect* con)</code>
<b>Parameters</b>	<code>--&gt; con</code> Pointer to <a href="#">IrConnect</a> structure.
<b>Result</b>	Returns nothing.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## **IrSetDeviceInfo**

<b>Purpose</b>	Sets the XID info string used during discovery to the given string and length.
<b>Declared In</b>	<code>IrLib.h</code>
<b>Prototype</b>	<code>IrStatus IrSetDeviceInfo (UInt16 refNum,                   UInt8 *info, UInt8 len)</code>
<b>Parameters</b>	<code>--&gt; refnum</code> IR library refNum. <code>--&gt; info</code> Pointer to array of bytes. <code>--&gt; len</code> Number of bytes pointed to by info.
<b>Result</b>	<code>IR_STATUS_SUCCESS</code> means the operation is successful. <code>IR_STATUS_FAILED</code> means the operation failed because info is too big.
<b>Comments</b>	The XID info string contains hints and the nickname of the device. The size cannot exceed <code>IR_MAX_DEVICE_INFO</code> bytes.

## IR Library

### *IR Library Functions*

---

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

## **IrTestReq**

**Purpose** Requests a TEST command frame be sent in the NDM (Normal Disconnect Mode) state.

**Declared In** IrLib.h

**Prototype** IrStatus IrTestReq (UInt16 refNum,  
IrDeviceAddr devAddr, IrConnect\* con,  
IrPacket\* packet)

**Parameters**

--> refnum	IR library refNum.
--> devAddr	Address of device where TEST is sent. This address is not checked so it can be the broadcast address or 0.
--> con	Pointer to <a href="#">IrConnect</a> structure specifying the callback function to call to report the result.
--> packet	Pointer to an <a href="#">IrPacket</a> structure that contains the data to send in the TEST command packet. The maximum size data that can be sent is IR_MAX_TEST_PACKET. Even if no data is to be sent, a valid packet must be passed.

**Result** IR\_STATUS\_PENDING means the operation has been started successfully and the result is returned by way of the callback function with the event LEVENT\_TEST\_CNF. This is also the indication returning the packet.

IR\_STATUS\_FAILED means the operation failed because of one of the following reasons. Note that the packet is available immediately.

- The IrConnect structure is not bound to the stack.
- The packet size exceeds the maximum allowed.

IR\_STATUS\_MEDIA\_BUSY means the operation failed because the media is busy or the stack is not in the NDM state (the packet is available immediately).

<b>Comments</b>	The result is signaled by way of the callback specified in the IrConnect structure. The callback event is LEVENT_TEST_CNF and the status field indicates the result of the operation. IR_STATUS_SUCCESS indicates success and IR_STATUS_FAILED indicates no response was received. A packet must be passed containing the data to send in the TEST frame. The packet is returned when the LEVENT_TEST_CNF event is given.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## IrUnbind

<b>Purpose</b>	Unbinds the IrConnect structure from the protocol stack, freeing its LSAP selector.
<b>Declared In</b>	<code>IrLib.h</code>
<b>Prototype</b>	<code>IrStatus IrUnbind (UInt16 refNum, IrConnect* con)</code>
<b>Parameters</b>	<code>--&gt; refnum</code> IR library refNum. <code>--&gt; con</code> Pointer to <a href="#">IrConnect</a> structure to unbind.
<b>Result</b>	IR_STATUS_SUCCESS means the operation completed successfully. IR_STATUS_FAILED means the operation failed for one of the following reasons: <ul style="list-style-type: none"><li>• The IrConnect structure was not bound.</li><li>• The lLsap field contained an invalid number.</li></ul>
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## IAS Functions

This section describes the following functions and macros related to IAS database:

- [IrIAS\\_Add](#)
- [IrIAS\\_GetInteger](#)

- [IriIAS\\_GetIntLsap](#)
- [IriIAS\\_GetObjectID](#)
- [IriIAS\\_GetOctetString](#)
- [IriIAS\\_GetOctetStringLen](#)
- [IriIAS\\_GetType](#)
- [IriIAS GetUserString](#)
- [IriIAS GetUserStringCharSet](#)
- [IriIAS GetUserStringLen](#)
- [IriIAS\\_Next](#)
- [IriIAS\\_Query](#)
- [IriIAS\\_SetDeviceName](#)
- [IriIAS\\_StartResult](#)

## **IriIAS\_Add**

**Purpose** Adds an IAS object to the IAS Database.

**Declared In** IrLib.h

**Prototype** IrStatus IriIAS\_Add (UInt16 refNum,  
IriiasObject\* obj)

**Parameters** --> refnum            IR library refNum.  
              --> obj              Pointer to an [IriIASObject](#) structure.

**Result** IR\_STATUS\_SUCCESS means the operation is successful.  
IR\_STATUS\_FAILED means the operation failed for one of the following reasons:

- There is no space in the database.
- An entry with the same class name already exists.
- The attributes of the object violate the IrDA Lite rules (attribute name exceeds IR\_MAX\_IAS\_NAME, or attribute value exceeds IR\_MAX\_IAS\_ATTR\_SIZE).

- The class name exceeds `IR_MAX_IAS_NAME`.

**Comments** The object is not copied, so the memory for the object must exist for as long as the object is in the database. The IAS database is designed to allow only objects with unique class names, and it checks for this. Class names and attributes names must not exceed `IR_MAX_IAS_NAME`. Also, attribute values must not exceed `IR_MAX_IAS_ATTR_SIZE`.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

## IrlIAS\_GetInteger

**Purpose** Macro to return an integer value, assuming that the current result item is of type `IAS_ATTRIB_INTEGER`. (Call [IrlIAS\\_GetType](#) to determine the type of the current result item.)

**Declared In** `IrlLib.h`

**Prototype** `IrlIAS_GetInteger (t)`

**Parameters** `--> t` Pointer to an [IrlIasQuery](#) structure.

**Result** Integer value returned as a `UInt32`.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

## IrlIAS\_GetIntLsap

**Purpose** Macro to return an integer value that represents an LSAP, assuming that the current result item is of type `IAS_ATTRIB_INTEGER`. (Call

[IrIAS\\_GetType](#) to determine the type of the current result item.)  
Usually integer values returned in a query are LSAP selectors.

**Declared In** IrLib.h

**Prototype** IrIAS\_GetIntLsap (t)

**Parameters** --> t                    Pointer to an [IrIasQuery](#) structure.

**Result** Integer value returned as a UInt8.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

## **IrIAS\_GetObjectID**

**Purpose** Macro to return the unique object ID of the current result item.

**Declared In** IrLib.h

**Prototype** IrIAS\_GetObjectID (t)

**Parameters** --> t                    Pointer to an [IrIasQuery](#) structure.

**Result** Returns the object ID as a UInt16 type.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

## IrIAS\_GetOctetString

<b>Purpose</b>	Macro to return a pointer to an octet string, assuming that the current result item is of type IAS_ATTRIB_OCTET_STRING. (Call <a href="#">IrIAS_GetType</a> to determine the type of the current result item.)
<b>Declared In</b>	IrLib.h
<b>Prototype</b>	<code>IrIAS_GetOctetString (t)</code>
<b>Parameters</b>	--> t Pointer to an <a href="#">IrIasQuery</a> structure.
<b>Result</b>	Pointer to octet string of type UInt8.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## IrIAS\_GetOctetStringLen

<b>Purpose</b>	Macro to return the length of an octet string, assuming that the current result item is of type IAS_ATTRIB_OCTET_STRING. (Call <a href="#">IrIAS_GetType</a> to determine the type of the current result item.)
<b>Declared In</b>	IrLib.h
<b>Prototype</b>	<code>IrIAS_GetOctetStringLen (t)</code>
<b>Parameters</b>	--> t Pointer to an <a href="#">IrIasQuery</a> structure.
<b>Result</b>	Length of octet string returned as a UInt16.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## IrlIAS\_GetType

<b>Purpose</b>	Macro to return the type of the current result item.
<b>Declared In</b>	<code>IrLib.h</code>
<b>Prototype</b>	<code>IrlIAS_GetType (t)</code>
<b>Parameters</b>	<code>--&gt; t</code> Pointer to an <a href="#">IrlIasQuery</a> structure.
<b>Result</b>	Type of result item, such as <code>IAS_ATTRIB_INTEGER</code> , <code>IAS_ATTRIB_OCTET_STRING</code> or <code>IAS_ATTRIB_USER_STRING</code> . The return value is of type <code>UInt8</code> .
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## IrlIAS\_GetUserString

<b>Purpose</b>	Macro to return a pointer to a user string, assuming that the current result item is of type <code>IAS_ATTRIB_USER_STRING</code> . (Call <a href="#">IrlIAS_GetType</a> to determine the type of the current result item.)
<b>Declared In</b>	<code>IrLib.h</code>
<b>Prototype</b>	<code>IrlIAS.GetUserString (t)</code>
<b>Parameters</b>	<code>--&gt; t</code> Pointer to an <a href="#">IrlIasQuery</a> structure.
<b>Result</b>	Pointer to result string of type <code>UInt8</code> .
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## IrlIAS.GetUserStringCharSet

<b>Purpose</b>	Macro to return the character set of the user string, assuming that the current result item is of type <code>IAS_ATTRIB_USER_STRING</code> .
----------------	--

(Call [IrIAS\\_GetType](#) to determine the type of the current result item.)

**Declared In** IrLib.h

**Prototype** IrIAS.GetUserStringCharSet (t)

**Parameters** --> t                    Pointer to an [IrIasQuery](#) structure.

**Result** Character set returned as an IrCharSet value.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

## **IrIAS.GetUserStringLen**

**Purpose** Macro to return the length of a user string, assuming that the current result item is of type IAS\_ATTRIB\_USER\_STRING. (Call [IrIAS\\_GetType](#) to determine the type of the current result item.)

**Declared In** IrLib.h

**Prototype** IrIAS.GetUserStringLen (t)

**Parameters** --> t                    Pointer to an [IrIasQuery](#) structure.

**Result** Length of user string returned as a UInt8 value.

**Compatibility** Implemented only if [3.0 New Feature Set](#) is present.

## IriIAS\_Next

<b>Purpose</b>	Moves the internal pointer to the next result item.
<b>Declared In</b>	<code>IrLib.h</code>
<b>Prototype</b>	<code>UInt8* IriIAS_Next (UInt16 refNum, IriIasQuery* token)</code>
<b>Parameters</b>	<code>--&gt; refnum</code> IR library refNum. <code>--&gt; token</code> Pointer to an <a href="#">IriIasQuery</a> structure.
<b>Result</b>	Pointer to the next result item, or 0 if there are no more items.
<b>Comments</b>	This function returns a pointer to the start of the next result item. If the pointer is 0, then there are no more result items.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## IriIAS\_Query

<b>Purpose</b>	Makes an IAS query of the IAS database of another device.
<b>Declared In</b>	<code>IrLib.h</code>
<b>Prototype</b>	<code>IrStatus IriIAS_Query (UInt16 refNum, IriIasQuery* token)</code>
<b>Parameters</b>	<code>--&gt; refnum</code> IR library refNum. <code>--&gt; token</code> Pointer to an <a href="#">IriIasQuery</a> structure initialized as described in the Comments section.
<b>Result</b>	<code>IR_STATUS_SUCCESS</code> means the operation is started successfully and the result is signaled by way of the callback function. <code>IR_STATUS_FAILED</code> means the operation failed for one of the following reasons:

- The query exceeds `IR_MAX_QUERY_LEN`.
- The `result` field of `token` is 0.
- The `resultBufSize` field of `token` is 0.
- The `callback` field of `token` is 0.
- A query is already in progress.

`IR_STATUS_NO_IRLAP` means the operation failed because there is no IrLAP connection.

**Comments**

An IrLAP connection must exist to the other device. The IAS query token must be initialized as described below. The result is signaled by calling the callback function whose pointer exists in the `IrIasQuery` structure. Only one query can be made at a time.

The `IrIasQuery` structure passed in the `token` parameter must be initialized as follows:

- Assign a pointer to a callback function in which the result is signaled.
- Set `result` to point to a buffer large enough to hold the result of the query.
- Set `resultBufSize` to the size of the result buffer.
- Set `queryBuf` to point to a valid query.
- Set `queryLen` to the number of bytes in `queryBuf`. The length must not exceed `IR_MAX_QUERY_LEN`.

**Compatibility**

Implemented only if [3.0 New Feature Set](#) is present.

## IrIAS\_SetDeviceName

<b>Purpose</b>	Sets the value field of the device name attribute of the “Device” object in the IAS database.						
<b>Declared In</b>	<code>IrLib.h</code>						
<b>Prototype</b>	<code>IrStatus IrIAS_SetDeviceName (UInt16 refNum,                   UInt8 *name, UInt8 len)</code>						
<b>Parameters</b>	<table><tr><td><code>--&gt; refnum</code></td><td>IR library refNum.</td></tr><tr><td><code>--&gt; name</code></td><td>Pointer to an IAS value field for the device name attribute of the device object. It includes the attribute type, character set and device name. This value field should be a constant and the pointer must remain valid until <code>IrIAS_SetDeviceName</code> is called with another pointer.</td></tr><tr><td><code>--&gt; len</code></td><td>Total length of the value field. Maximum size allowed is <code>IR_MAX_IAS_ATTR_SIZE</code>.</td></tr></table>	<code>--&gt; refnum</code>	IR library refNum.	<code>--&gt; name</code>	Pointer to an IAS value field for the device name attribute of the device object. It includes the attribute type, character set and device name. This value field should be a constant and the pointer must remain valid until <code>IrIAS_SetDeviceName</code> is called with another pointer.	<code>--&gt; len</code>	Total length of the value field. Maximum size allowed is <code>IR_MAX_IAS_ATTR_SIZE</code> .
<code>--&gt; refnum</code>	IR library refNum.						
<code>--&gt; name</code>	Pointer to an IAS value field for the device name attribute of the device object. It includes the attribute type, character set and device name. This value field should be a constant and the pointer must remain valid until <code>IrIAS_SetDeviceName</code> is called with another pointer.						
<code>--&gt; len</code>	Total length of the value field. Maximum size allowed is <code>IR_MAX_IAS_ATTR_SIZE</code> .						
<b>Result</b>	<code>IR_STATUS_SUCCESS</code> means the operation is successful. <code>IR_STATUS_FAILED</code> means <code>len</code> is too big, or the value field is not a valid user string.						
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.						

## IrlAS\_StartResult

<b>Purpose</b>	Macro to put the internal pointer to the start of the result buffer.
<b>Declared In</b>	IrLib.h
<b>Prototype</b>	<code>IrlAS_StartResult (t)</code>
<b>Parameters</b>	<code>--&gt; t</code> Pointer to an <a href="#">IrIasQuery</a> structure.
<b>Result</b>	Returns nothing.
<b>Compatibility</b>	Implemented only if <a href="#">3.0 New Feature Set</a> is present.

## Application-Defined Functions

The functions in this section are supplied by the developer and can be named anything.

### IrlasQueryCallBack

<b>Purpose</b>	Signals the result of IAS query. The result of IAS queries is signaled by calling this callback function which is pointed to by the callBack field of the <a href="#">IrIasQuery</a> structure.
<b>Declared In</b>	IrLib.h
<b>Prototype</b>	<code>void IrIasQueryCallBack (IrStatus status)</code>
<b>Parameters</b>	<code>--&gt; status</code> The status of the query operation. The following values can be passed:  IR_STATUS_SUCCESS means the query operation finished successfully and the results can be parsed.  IR_STATUS_DISCONNECT means the link or IrLMP connection was disconnected

## **IR Library**

### *Application-Defined Functions*

---

during the query, so the results are not valid.

**Result** Returns nothing.

# Modem Manager

---

This chapter provides reference material for the modem manager API. The header file ModemMgr.h declares the modem manager API.

## Modem Manager Functions

### MdmDial

**Purpose** Initialize the modem, dial the phone number and wait for result.

**Declared In** ModemMgr.h

**Prototype** Err MdmDial (MdmInfoPtr modemP, Char \*okDialP,  
Char \*userInitP, Char \*phoneNumP)

<b>Parameters</b>	modemP	Pointer to modem info structure (filled in by caller)
	okDialP	(NOT IMPLEMENTED) Pointer to string of chars allowed in dial string
	userInitP	Pointer to modem setup string without the AT prefix.
	phoneNumP	Pointer to phone number string

**Result** 0 if successful; otherwise mdmErrNoTone, mdmErrNoDCD,  
mdmErrBusy, mdmErrUserCan, mdmErrCmdError

**Comments** When executing this function, the system performs these steps:

- Switch to the requested initial baud rate.
- If HW hand-shake is requested, enable CTS/RTS hand-shaking; otherwise, disable it.

## **Modem Manager**

### *Modem Manager Functions*

---

- Reset the modem.
- Execute the setup string (if any).
- Configure the modem with required settings.
- Dial the phone number.
- Wait for CONNECT XXXXX or other response.
- If auto-baud is requested, switch to the connected baud rate.

## **MdmHangUp**

**Purpose** Hang up the modem.

**Declared In** ModemMgr.h

**Prototype** Err MdmHangUp (MdmInfoPtr modemP)

**Parameters** modemP Pointer to modem info structure (filled in by caller)

**Result** 0 if successful.

---

**WARNING!** This function alters configuration of the serial port (without restoring it).

---

# Net Library

---

This chapter describes the API available in the net library and its Berkeley sockets equivalents. The header file `NetMgr.h` declares the net library API. The chapter covers:

- [Net Library Data Structures](#)
- [Net Library Constants](#)
- [Net Library Functions](#)

For more information on the net library, see the chapter “[Network Communication](#)” in the *Palm OS Programmer’s Companion*, vol. II, *Communications*.

---

**IMPORTANT:** Applications cannot directly use the net library to make wireless connections. Use the INetLib for wireless connections.

---

## Net Library Data Structures



### NetConfigNameType

The `NetConfigNameType` structure defines a configuration name. A configuration is a specific set of values for the net library settings. Typically, users set up configurations and assign names to them using the Network preferences panel.

```
typedef struct {
    Char name[netConfigNameSize];
} NetConfigNameType, NetConfigNamePtr;
```

`name` is the configuration’s name. The `netConfigNameSize` constant is currently defined to be 32.

### Compatibility

Supported only if [3.2 New Feature Set](#) is present.

## NetHostInfoBufType

The NetHostInfoBufType struct contains information about a host. The [NetHostInfoType](#) struct, which maps to the hostent struct, points to fields in this struct for its information.

```
typedef struct {
    NetHostInfoType hostInfo;
    Char           name [netDNSMaxDomainName+1];
    Char           *aliasList [netDNSMaxAliases+1];
    Char           aliases [netDNSMaxAliases]
                    [netDNSMaxAliases+1];
    NetIPAddr     *addressList [netDNSMaxAddresses];
;
    NetIPAddr     address [netDNSMaxAddresses];
} NetHostInfoBufType, *NetHostInfoBufPtr;
```

### Field Descriptions

hostInfo	A <a href="#">NetHostInfoType</a> struct, which maps to the Berkeley UNIX sockets hostent structure.
name	Official host name.
aliasList	An array of aliases for the host name.
aliases	
addressList	An array of pointers to 32-bit IP addresses in host byte order.
address	

## NetHostInfoType

The NetHostInfoType structure maps to the Berkeley UNIX sockets hostent structure. It is defined as follows:

```
typedef struct {
    Char           *nameP;
    Char           **nameAliasesP;
    UInt16         addrType;
    UInt16         addrLen;
```

```
    UInt8      **addrListP;
} NetHostInfoType, *NetHostInfoPtr;
```

### Field Descriptions

nameP	Official host name.
nameAliasesP	An array of aliases for the host name.
addrType	The type of the return addresses. See <a href="#"><u>NetSocketAddrEnum</u></a> .
addrLen	The length in bytes of the return addresses.
addrListP	An array of pointers to addresses in host byte order.

## NetServInfoBufType

The NetServInfoBufType struct contains information about a service. The [NetServInfoType](#) struct, which maps to the servent struct, points to fields in this struct for much of its information.

```
struct {
    NetServInfoType servInfo;
    Char           name [netServMaxName+1];
    Char           *aliasList [netServMaxAliases+1];
    Char           aliases [netServMaxAliases]
    [netServMaxName];
    Char           protoName [netProtoMaxName+1];
    UInt8          reserved;
} NetServInfoBufType, *NetServInfoBufPtr;
```

### Field Descriptions

servInfo	A <a href="#">NetServInfoType</a> struct, which maps to the Berkeley UNIX sockets servent structure.
name	Official name of the service
aliasList	Array of aliases for the service name.
aliases	
protoName	Name of the protocol to use.
reserved	Reserved for future use.

## NetServInfoType

The NetServInfoType structure maps to the servent structure in Berkeley UNIX sockets API. It contains information about a service.

```
struct {
    Char      *nameP;
    Char      **nameAliasesP;
    UInt16    port;
    Char      *protoP;
} NetServInfoType, *NetServInfoPtr;
```

### Field Descriptions

nameP	Official name of the service
nameAliasesP	Array of aliases for the service name.
port	Port number for the service.
protoP	Name of the protocol to use.

### NetSocketAddrEnum

The NetSocketAddrEnum enum specifies the address types supported by the net library.

```
typedef enum {
    netSocketAddrRaw = 0,
    netSocketAddrINET = 2
} NetSocketAddrEnum
```

### Value Descriptions

netSocketAddrRaw	Raw address. Supported in Palm OS® version 3.0 and higher.
netSocketAddrINET	IP address.

### NetSocketAddrINType

The NetSocketAddrINType struct holds an internet socket address, that is, a socket that uses one of the internet protocols. This structure directly maps to the BSD UNIX sockaddr\_in structure.

```
typedef struct NetSocketAddrINType {
    Int16      family;
    UInt16     port;
    NetIPAddr  addr;
} NetSocketAddrINType;
```

## Net Library

### Net Library Data Structures

---

#### Field Descriptions

- family Address family in host byte order. This is either netSocketAddrINET or netSocketAddrRaw.
- port The port in network byte order.
- addr The IP address in network byte order.

### NetSocketAddrRawType

The NetSocketAddrRawType structure holds a raw socket address.

```
typedef struct NetSocketAddrRawType {  
    Int16 family;  
    UInt16 ifInstance;  
    UInt32 ifCreator;  
} NetSocketAddrRawType;
```

#### Field Descriptions

- family Address family in host byte order. This is either netSocketAddrINET or netSocketAddrRaw.
- ifInstance The instance number of the interface that the socket uses to send and receive data.
- ifCreator The creator of the interface that the socket uses.

**Compatibility** Raw sockets are supported in Palm OS version 3.0 and higher.

## NetSocketAddrType

The `NetSocketAddrType` structure holds a generic socket address. This struct can hold any type of address including Internet addresses. It directly maps to the BSD UNIX `sockaddr` structure.

Note that this structure is the same size as `NetSocketAddrINType` and `NetSocketAddrRawType`. This means that one of those two structures can be used for parameters declared to be `NetSocketAddrType`.

```
typedef struct NetSocketAddrType {  
    Int16 family;  
    UInt8 data[14];  
} NetSocketAddrType;
```

## NetSocketRef

The `NetSocketRef` defines a socket descriptor. The socket descriptor is created and returned by [NetLibSocketOpen](#). It is used in any function that requires access to a socket.

```
typedef Int16 NetSocketRef
```

## NetSocketTypeEnum

The `NetSocketTypeEnum` enum specifies the available socket types.

```
typedef enum {  
    netSocketTypeStream=1,  
    netSocketTypeDatagram=2,  
    netSocketTypeRaw=3,  
    netSocketTypeReliableMsg=4  
} NetSocketTypeEnum
```

### Value Descriptions

<code>netSocketTypeStream</code>	Streams protocol over wireline.
<code>netSocketTypeDatagram</code>	UDP protocol.
<code>netSocketTypeRaw</code>	Raw mode.

# Net Library Constants



## Configuration Aliases

A configuration is a set of specific values for the net library settings. The net library defines a set of built-in configuration aliases for common network setups. These aliases point to configurations instead of holding the actual values themselves. You can specify an alias anywhere in the API you would specify a configuration.

The constants listed here specify the alias names. Most of the net library API requires a configuration index rather than a name. Use [NetLibConfigIndexFromName](#) to obtain the alias's index from the name.

<code>netCfgNameDefault</code>	The default configuration.
<code>netCfgNameDefWireline</code>	The default configuration for wireline communications.
<code>netCfgNameDefWireless</code>	The default configuration for wireless communications.
<code>netCfgNameCTPWireline</code>	The default configuration for wireline communications through the Palm Web Clipping Proxy server.
<code>netCfgNameCTPWireless</code>	The default configuration for wireless communications through the Palm Web Clipping Proxy server.

By default, `netCfgNameDefault` points to the user's default configuration, and all other aliases point to `netCfgNameDefault` except for `netCfgNameCTPWireless`, which points to an private wireless configuration.

### Compatibility

Supported on version 3.2 and later.

## I/O Flags

The I/O flags specify special handling instructions to functions that send and receive data. You can OR these values together to specify more than one.

netIOFlagOutOfBand	Process out-of-band data. Available for send calls only.
netIOFlagPeek	Peek at incoming message without dequeuing it.
netIOFlagDontRoute	Send without using routing. This constant is currently ignored.

## Tracing Bits

The tracing bits are used to set the level of event tracing. An application can get a list of events in the trace buffer using the [NetLibMaster](#) call.

You can set the tracing for each network interface using [NetLibIFSettingSet](#) and for the net library in general with [NetLibSettingSet](#).

netTracingErrors	Record run-time errors. This is the default.
netTracingMsgs	Record application trace messages.
netTracingPkts	Record packet I/O. This bit is obsolete in versions 3.2 and higher, but is mapped to netTracingPktIP.
netTracingFuncs	Record function flow.
netTracingAppMsgs	Record application messages sent using <a href="#">NetLibTracePrintF</a> and <a href="#">NetLibTracePutS</a> .
netTracingPktIP	Record packet I/O. If this set, the following five options are enabled.

netTracingData40	Record the first 40 bytes of each packet sent or received. This option is mutually exclusive with netTracingData.
netTracingData	Record the entirety of each packet sent or received. This option is mutually exclusive with netTracingData40.
netTracingIFHi	Record packets sent or received at the highest layer of the network interface. This layer is just below the IP layer.
netTracingIFMid	Record packets sent or received at the layer just below the highest layer of the network interface.
netTracingIFLow	Record packets sent or received at the lowest layer of the network interface.

### **Compatibility**

The netTracingPktXXX constants are supported only in version 3.2 devices and higher. In previous versions, specify netTracingPkts instead; only the size of the packet is recorded.

## **Net Library Functions**

### **NetHToNL**

<b>Purpose</b>	Macro that converts a 32-bit value from host to network byte order.
<b>Declared In</b>	NetBitUtils.h
<b>Prototype</b>	NetHToNL (x)
<b>Parameters</b>	-> x                            32-bit value to convert.
<b>Result</b>	Returns x in network byte order.

**Sockets  
Equivalent** htonl()

**See Also** [NetNToHS](#), [NetNToHL](#), [NetHToNS](#)

## NetHToNS

**Purpose** Macro that converts a 16-bit value from host to network byte order.

**Declared In** NetBitUtils.h

**Prototype** NetHToNS (x)

**Parameters** -> x 16-bit value to convert.

**Result** Returns x in network byte order.

**Sockets  
Equivalent** htons()

**See Also** [NetNToHS](#), [NetNToHL](#), [NetHToNL](#)

## NetLibAddrAToIN

**Purpose** Converts an ASCII string representing a dotted decimal IP address into a 32-bit IP address in network byte order.

**Declared In** NetMgr.h

**Prototype** NetIPAddr NetLibAddrAToIN (UInt16 libRefnum,  
const Char \*a)

**Parameters** -> libRefNum Reference number of the net library.  
-> a Pointer to ASCII dotted decimal string.

**Result** Returns a 32-bit network byte order IP address or -1 if a doesn't represent a dotted decimal IP address

**Sockets Equivalent**    `UInt32 inet_addr (char *cp)`

**See Also**    [NetLibAddrINToA](#)

## NetLibAddrINToA

**Purpose**    Converts an IP address from 32-bit network byte order into a dotted decimal ASCII string.

**Declared In**    `NetMgr.h`

**Prototype**    `Char *NetLibAddrINToA (UInt16 libRefnum,  
NetIPAddr inet, Char *spaceP)`

**Parameters**    -> `libRefNum`    Reference number of the net library.  
-> `inet`    32-bit IP address in network byte order.  
<- `spaceP`    Buffer used to hold the return value.

**Result**    Returns in `spaceP` the dotted decimal ASCII string representation of the IP address.

**Sockets Equivalent**    `char *inet_ntoa (struct in_addr in)`

**See Also**    [NetLibAddrATOIN](#)

## NetLibClose

**Purpose**    Closes the net library.

**Declared In**    `NetMgr.h`

**Prototype**    `Err NetLibClose (UInt16 libRefnum,  
UInt16 immediate)`

**Parameters**    -> `libRefnum`    Reference number of the net library.

-> `immediate` If `true`, library will shut down immediately. If `false`, library will shut down only if close timer expires before another [NetLibOpen](#) is issued.

**Result** Returns one of the following values:

0 Success.

`netErrNotOpen` Library was not open.

`netErrStillOpen`

Not really an error; returned if library is still in use by another application.

**Sockets  
Equivalent** None.

**Comments** Applications must call this function when they no longer need the net library. If the net library open count is greater than 1 before this call is made, the count is decremented and `netErrStillOpen` is returned. If the open count was 1, the library takes the following action:

- If `immediate` is `true`, the library shuts down immediately. All network interfaces are brought down, the net protocol stack task is terminated, and all memory used by the net library is freed.
- If `immediate` is `false`, a close timer is created and this call returns immediately without actually bringing the net library down. Instead it leaves it up and running but marks it as in the “close-wait” state. It remains in this state until either the timer expires or another `NetLibOpen` is issued. If the timer expires, the library is shut down. If another `NetLibOpen` call is issued before the timer expires (possibly by another application), the timer is cancelled and the library is marked as fully open.

In most cases, you should pass `false` for `immediate`. This allows the user to quit one Internet application and launch another within

a short period of time without having to wait through the process of closing down and then re-establishing dial-up network connections.

**See Also** [NetLibOpen](#), [NetLibOpenCount](#)



## NetLibConfigAliasGet

**Purpose** Return the configuration that an alias points to.

**Prototype** Err NetLibConfigAliasGet (UInt16 refNum,  
                  UInt16 aliasIndex, UInt16 \*indexP,  
                  Boolean \*isAnotherAliasP)

**Parameters**

-> refNum	Reference number of the net library.
-> aliasIndex	Index of the alias.
<- indexP	Index of the configuration pointed to by the alias.
<- isAnotherAliasP	true if indexP is the index of another alias; false if indexP specifies an actual configuration.

**Result** Returns one of the following values:

0 Success.

netErrConfigNotAlias

The configuration at aliasIndex is not an alias.

netErrOutOfCmdBlocks

netErrParamErr The specified index is out of range or there is no configuration at the index.

**Sockets Equivalent** None

<b>Comments</b>	Use this routine to find out which configuration a built-in alias points to. See “ <a href="#">Configuration Aliases</a> ” for a description of the built-in aliases.
<b>Compatibility</b>	Implemented only if <a href="#">3.2 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">NetLibConfigAliasSet</a>

 **New**

## **NetLibConfigAliasSet**

<b>Purpose</b>	Set a built-in alias to point to a defined configuration.						
<b>Prototype</b>	<code>Err NetLibConfigAliasSet (UInt16 refNum,                   UInt16 configIndex, UInt16 aliasToIndex)</code>						
<b>Parameters</b>	<table><tr><td>-&gt; refNum</td><td>Reference number of the net library.</td></tr><tr><td>-&gt; configIndex</td><td>Index of the built-in alias to be set.</td></tr><tr><td>-&gt; aliasToIndex</td><td>Index of the configuration to which the alias should point. You cannot set an alias to point to itself.</td></tr></table>	-> refNum	Reference number of the net library.	-> configIndex	Index of the built-in alias to be set.	-> aliasToIndex	Index of the configuration to which the alias should point. You cannot set an alias to point to itself.
-> refNum	Reference number of the net library.						
-> configIndex	Index of the built-in alias to be set.						
-> aliasToIndex	Index of the configuration to which the alias should point. You cannot set an alias to point to itself.						
<b>Result</b>	Returns one of the following values:  0 Success.  <code>netErrConfigCantPointToAlias</code> The configuration at aliasToIndex is an alias that points to an alias.  <code>netErrConfigNotAlias</code> The configuration at configIndex isn't an alias.  <code>netErrOutOfCmdBlocks</code>  <code>netErrParamErr</code> The specified index is out of range or there's no configuration at the index.						

<b>Sockets Equivalent</b>	None
<b>Comments</b>	<p>This function is used by the Network preferences panel when the user edits a configuration. Your application can use it to associate any of the built-in aliases with a defined configuration.</p> <p>The built-in aliases are typically set up as shown in <a href="#">Table 61.1</a>. In this example, applications that specify a configuration index of 0 through 3 use a configuration that the user defines. Applications that use index 4 use a private configuration created by the network library.</p>

**Table 61.1 Example Configuration Table**

Index	Name	Alias To
0	.Default	6
1	.DefWireline	0
2	.DefWireless	0
3	.CTPWireline	0
4	.CTPWireless	5
5	_RAMCTP	
6	user-defined	

An alias can point to another alias so long as the nesting level is only one deep. That is, if you point an alias to an alias, you'll receive an error if that alias in turn points to another alias. This eliminates the possibility that an alias never resolves to an actual configuration.

**Compatibility** Implemented only if [3.2 New Feature Set](#) is present.

**See Also** [NetLibConfigAliasGet](#)



## New **NetLibConfigDelete**

**Purpose** Delete a configuration from the net library's configuration table.

**Prototype** Err NetLibConfigDelete (UInt16 refNum,  
UInt16 index)

**Parameters**

- > refNum	Reference number of the net library.
- > index	Index of the configuration to delete. You cannot delete one of the built-in aliases described in " <a href="#">Configuration Aliases</a> ."

**Result** Returns one of the following values:

0 Success.

netErrConfigCantDelete

The configuration at index is a built-in alias.

netErrOutOfCmdBlocks

netErrParamErr The specified index is out of range.

**Sockets  
Equivalent**

**Compatibility** Implemented only if [3.2 New Feature Set](#) Set is present.

**See Also** [NetLibConfigSaveAs](#)



## New **NetLibConfigIndexFromName**

**Purpose** Obtain a configuration's index given its name.

**Prototype** Err NetLibConfigIndexFromName (UInt16 refNum,  
NetConfigNamePtr nameP, UInt16 \*indexP)

**Parameters**

-> refNum	Reference number of the net library.
-> nameP	Pointer to a configuration name. See <a href="#">NetConfigNameType</a> .
<- indexP	The index of the configuration with the name *nameP.

**Result** Returns one of the following values:

0 Success.

netErrConfigNotFound A configuration with the specified name could  
not be found.

netErrOutOfCmdBlocks

**Sockets  
Equivalent** None

**Comments** This function returns the index of a configuration given its name.  
Your application should store the configuration's index rather than  
its name because a configuration's name can change.

If you pass the name of a built-in alias in nameP, this function  
returns the index of the alias's entry in the configuration table; it  
does not return the index that the alias points to. For example, if the  
alias netCfgNameCTPWireless is stored at index 4 and points to  
index 5, NetLibConfigIndexFromName returns 4. If you want to  
obtain the index that an alias points to, use  
[NetLibConfigAliasGet](#).

**Compatibility** Implemented only if [3.2 New Feature Set](#) is present.

**See Also** [NetLibConfigList](#)



## **NetLibConfigList**

**Purpose** Return a list of net library configuration names.

**Prototype** Err NetLibConfigList (UInt16 refNum,  
NetConfigNameType nameArray[],  
UInt16 \*arrayEntriesP)

**Parameters**

-> refNum	Reference number of the net library.
<- nameArray	The list of defined configurations. See <a href="#">NetConfigNameType</a> .
<-> arrayEntriesP	On entry, contains the number of elements in nameArray. On return, contains the number of elements in nameArray that were actually used. The Net Library currently returns up to 16 entries. If the array is not large enough to hold all the configuration names, this function returns only as many names as the array can hold.

**Result** Returns one of the following values:

0 Success.

netErrOutOfCmdBlocks

**Sockets  
Equivalent** None

**Comments** Use this function to obtain a list of the names of defined network configurations and configuration aliases.

Users create specific configurations using the Network preferences panel and associate names with each configuration. This function returns the list of defined configurations.

In addition to user-defined configurations, this function also returns built-in configuration aliases and private configurations. The built-in configuration aliases are described in “[Configuration Aliases](#).” Their actual names begin with a period (.). Private configurations have names that begin with an underscore (\_).

---

**IMPORTANT:** If you present the list returned by this function to your application’s users, you must first filter out names beginning with a period or an underscore. These names are for internal use only.

---

Your application should refer to a configuration by its index rather than its name because the name can be changed. To obtain the configuration’s index from its name, use [NetLibConfigIndexFromName](#).

**Compatibility** Implemented only if [3.2 New Feature Set](#) is present.



## NetLibConfigSetActive

**Purpose** Makes the specified configuration current without opening the net library.

**Prototype** Err NetLibConfigSetActive (UInt16 refNum,  
UInt16 configIndex)

**Parameters**

-> refNum	Reference number of the net library.
-> configIndex	Index of the configuration to use. An index of 0 refers to the default configuration as defined by the Network preferences panel.

**Result** Returns one of the following values:

0	Success.
	netErrBufTooSmall
	netErrConfigAliasErr
	netErrConfigCantDelete
	netErrConfigEmpty
	netErrConfigNotFound
	netErrOutOfCmdBlocks
	netErrParamErr
	netErrPrefNotFound

**Sockets  
Equivalent** None

**Comments** This function is used mainly by the Network preferences panel when the user edits and saves network configurations. The Network preferences panel uses this function to make current the configuration the user wants to edit, set the settings appropriately, and then save the configuration using [NetLibConfigSaveAs](#).  
Use this routine to make a specific configuration the current configuration without opening the net library. You should not use it if the net library is already open.  
Unlike [NetLibOpenConfig](#), this routine does not save the current net library configuration so that it can be restored upon close.

**Compatibility** Implemented only if [3.2 New Feature Set](#) is present.



## New **NetLibConfigRename**

**Purpose** Rename the specified configuration.

**Prototype** Err NetLibConfigRename (UInt16 refNum,  
                  UInt16 index, NetConfigNamePtr newNameP)

**Parameters**

-> refNum	Reference number of the net library.
-> configIndex	Index of the configuration to be renamed.
-> newNameP	Pointer to the new name. See <a href="#">NetConfigNameType</a> . The new name must not start with a period (.) or an underscore (_).

**Result** Returns one of the following values:

0	Success.
netErrConfigBadName	The new name begins with a period.
netErrConfigCantDelete	The configuration at the specified index is a built-in alias or private configuration that cannot be renamed.
netErrOutOfCmdBlocks	
netErrParamErr	The specified index is out of range or there is no configuration at the index.

**Sockets Equivalent** None

**Comments** You cannot specify a name beginning with a period (.) or an underscore (\_). Names beginning with a period are reserved for the built-in configuration aliases. Names beginning with an underscore are hidden configurations used internally by net library.

**Compatibility** Implemented only if [3.2 New Feature Set](#) is present.



## New **NetLibConfigSaveAs**

**Purpose** Save the current net library settings as a configuration with the specified name.

**Prototype** Err NetLibConfigSaveAs (UInt16 refNum,  
NetConfigNamePtr nameP)

**Parameters**

- > refNum	Reference number of the net library.
- > nameP	Pointer to a name for the configuration. See <a href="#">NetConfigNameType</a> . The name must not start with a period (.) or an underscore (_).

**Result** Returns one of the following values:

0 Success.

netErrConfigBadName

The specified name begins with a period or underscore.

netErrConfigTooMany

Not enough space to add another configuration. The Net Library can hold up to 16 configurations.

netErrOutOfCmdBlocks

**Sockets Equivalent** None

**Comments** If the name you specify already exists, its configuration is replaced with this configuration.

You cannot specify a name beginning with a period (.) or an underscore (\_). Names beginning with a period are reserved for the built-in configuration aliases. Names beginning with an underscore are hidden configurations used internally by net library.

The net library assigns an index to this new configuration. The configuration's index remains constant, while its name may change.

Use [NetLibConfigIndexFromName](#) to obtain the configuration's index.

**Compatibility** Implemented only if [3.2 New Feature Set](#) is present.

**See Also** [NetLibConfigDelete](#), [NetLibConfigRename](#)

## NetLibConnectionRefresh

**Purpose** This routine is a convenience call for applications. It checks the status of all connections and optionally tries to open any that were closed.

**Declared In** NetMgr.h

**Prototype** Err NetLibConnectionRefresh (UInt16 refNum,  
Boolean refresh, UInt8 \*allInterfacesUpP,  
UInt16 \*netIFErrP)

**Parameters**

-> refnum	Reference number of the net library.
-> refresh	If true, any connections that aren't currently open are opened.
<- allInterfacesUpP	Set to true if all connections are open.
<- netIFErrP	First error encountered when reopening connections that were closed. (See <a href="#">NetLibIFUp</a> for a list of possible values.)

**Result** Returns one of the following values:

0	Success.
	netErrBufTooSmall
	netErrOutOfCmdBlocks
	netErrNoInterfaces

**Sockets Equivalent** None.

**Comments** This function determines whether a connection is up based on the internal status of the TCP/IP stack. To test the presence of a “physical connection” (phone line, modem, serial cable), a command should be sent once it’s been determined that the logical connection is up. If the physical connection is broken, nothing returns and a timeout error eventually occurs.

## NetLibDmReceive

**Purpose** Receive data from a socket directly into a database record.

**Declared In** NetMgr.h

**Prototype** Int16 NetLibDmReceive (UInt16 libRefNum,  
NetSocketRef socket, void \*recordP,  
UInt32 recordOffset, UInt16 rcvLen, UInt16 flags,  
void \*fromAddrP, UInt16 \*fromLenP, Int32 timeout,  
Err \*errP)

**Parameters**

-> libRefNum	Reference number of the net library.
-> socket	Descriptor for the open socket.
<- recordP	Pointer to beginning of record to receive data into. Must be locked for use.
-> recordOffset	Offset from beginning of record to read data into.
-> rcvLen	Maximum number of bytes to read.
-> flags	One or more <i>netIOFlagxxx</i> flags. See “ <a href="#">I/O Flags</a> .”
<- fromAddrP	Pointer to buffer to hold address of sender (a <a href="#">NetSocketAddrType</a> struct). Pass NULL if you don’t need sender information.
<-> fromLenP	On entry, size of fromAddrP buffer. On exit, actual size of returned address in fromAddrP. Pass NULL if you don’t need sender information.

-> timeout      Maximum timeout in system ticks; -1 means wait forever.

<- errP      Contains an error code if the return value is -1.

**Result**      Returns the number of bytes successfully received. If the return value is 0, the socket has been shut down by the remote host. If the return value is -1, an error has occurred and errP contains one of the following values:

0      No error.

netErrTimeout      Call timed out.

netErrNotOpen      The referenced net library has not been opened yet.

netErrParamErr

netErrSocketNotOpen

netErrWouldBlock

netErrUserCancel

netErrOutOfMemory

**Comments**      This call behaves similarly to [NetLibReceive](#) but reads the data directly into a database record, which is normally write-protected. The caller must pass a pointer to the start of the record and an offset into the record of where to start the read.

## NetLibFinishCloseWait

**Purpose**      Forces the net library to do a complete close if it's currently in the close-wait state.

**Declared In**      NetMgr.h

**Prototype**      Err NetLibFinishCloseWait (UInt16 libRefnum)

**Parameters**      -> libRefnum      Reference number of the net library.

**Result**      Returns one of the following values:

<b>Sockets Equivalent</b>	0 netErrTimeout	Success.
<b>Comments</b>	None.	
<b>Comments</b>	This call checks the current open state of the net library. If it's in the close-wait state (see <a href="#">NetLibClose</a> ), it forces the library to perform an immediate, complete close operation.	
<b>NetLibGetHostByAddr</b>		
<b>Purpose</b>	Looks up a host name given its IP address.	
<b>Declared In</b>	NetMgr.h	
<b>Prototype</b>	NetHostInfoPtr NetLibGetHostByAddr (UInt16 libRefNum, UInt8 *addrP, UInt16 len, UInt16 type, NetHostInfoBufPtr bufP, Int32 timeout, Err *errP)	
<b>Parameters</b>	-> libRefNum	Reference number of the net library.
	-> addrP	IP address of host to lookup.
	-> len	Length, in bytes, of *addrP.
	-> type	Type of addrP. See <a href="#">NetSocketAddrEnum</a> .
	<- bufP	Pointer to a <a href="#">NetHostInfoBufType</a> struct in which to store the results of the lookup.
	-> timeout	Maximum timeout in system ticks; -1 means wait forever.
	<- errP	Contains an error code if the return value is 0.
<b>Result</b>	Returns a pointer to the <a href="#">NetHostInfoType</a> portion of bufP that contains results of the lookup. If the return value is 0, an error has occurred, and errP contains one of the following values:	
	0	No error

## Net Library

### Net Library Functions

---

netErrTimeout Call timed out.  
netErrNotOpen The referenced net library has not been opened yet.  
netErrDNSNameTooLong  
netErrDNSBadName  
netErrDNSLabelTooLong  
netErrDNSAllocationFailure  
netErrDNSTimeout  
netErrDNSUnreachable  
netErrDNSFormat  
netErrDNSServerFailure  
netErrDNSNonexistentName  
netErrDNSNIY  
netErrDNSRefused  
netErrDNSImpossible  
netErrDNSNoRRS  
netErrDNSAborted  
netErrDNSBadProtocol  
netErrDNSTruncated  
netErrDNSNoRecursion  
netErrDNSIrrelevant  
netErrDNSNotInLocalCache  
netErrDNSNoPort

**Sockets Equivalent** struct hostent \*gethostbyaddr (char \*addr,  
int len, int type);

**Comments** This call queries the domain name server(s) to look up a host name given its IP address.

**See Also** [NetLibGetHostByName](#)

## NetLibGetHostByName

**Purpose** Looks up a host IP address given a host name.

**Declared In** NetMgr.h

**Prototype** `NetHostInfoPtr NetLibGetHostByName  
(UInt16 libRefNum, const Char *nameP,  
NetHostInfoBufPtr bufP, Int32 timeout, Err *errP)`

**Parameters**

-> libRefNum	Reference number of the net library.
-> nameP	Name of host to look up.
<- bufP	Pointer to a <a href="#">NetHostInfoBufType</a> struct in which to store the results of the lookup.
-> timeout	Maximum timeout in system ticks; -1 means wait forever.
<- errP	Contains an error code if the return value is 0.

**Result** Returns a pointer to the [NetHostInfoType](#) portion of bufP, which contains results of the lookup. If the return value is 0, an error has occurred and errP contains one of the following values:

0	No error
netErrTimeout	Call timed out.
netErrNotOpen	The referenced net library has not been opened yet.
netErrDNSNameTooLong	
netErrDNSBadName	
netErrDNSLabelTooLong	
netErrDNSAllocationFailure	
netErrDNSTimeout	
netErrDNSUnreachable	
netErrDNSFormat	
netErrDNSServerFailure	

## Net Library

### Net Library Functions

---

```
netErrDNSNonexistentName  
netErrDNSNIY  
netErrDNSRefused  
netErrDNSImpossible  
netErrDNSNoRRS  
netErrDNSAborted  
netErrDNSBadProtocol  
netErrDNSTruncated  
netErrDNSNoRecursion  
netErrDNSIrrelevant  
netErrDNSNotInLocalCache  
netErrDNSNoPort
```

**Sockets Equivalent** struct hostent \*gethostbyname (char \*name);

**Comments** This call first checks the local name -> IP address host table in the net library preferences. If the entry is not found, it then queries the domain name server(s).

**See Also** [NetLibGetHostByAddr](#), [NetLibGetMailExchangeByName](#)

## NetLibGetMailExchangeByName

**Purpose** Looks up the name of a host to use for a given mail exchange.

**Declared In** NetMgr.h

**Prototype** Int16 NetLibGetMailExchangeByName  
(UInt16 libRefNum, Char \*mailNameP,  
 UInt16 maxEntries, Char hostNames[] [255+1],  
 UInt16 priorities[], Int32 timeout, Err \*errP)

**Parameters** -> libRefNum Reference number of the net library.

-> mailNameP	Name of the mail exchange to look up.
-> maxEntries	Maximum number of host names to return.
<- hostNames	Array of character strings of length 255+1. The host name results are stored in this array. This array must be able to hold at least maxEntries host names.
<- priorities	Array of Words. The priorities of each host name found are stored in this array. This array must be at least maxEntries in length.
-> timeout	Maximum timeout in system ticks; -1 means wait forever.
<- errP	Contains an error code if the return value is less than 0.

**Result** Returns the number of entries successfully found. If the return value is a negative number, an error has occurred, and `errP` contains one of the following values:

0	No error
netErrTimeout	Call timed out.
netErrNotOpen	The referenced net library has not been opened yet.
netErrDNSNameTooLong	
netErrDNSBadName	
netErrDNSLabelTooLong	
netErrDNSAllocationFailure	
netErrDNSTimeout	
netErrDNSUnreachable	
netErrDNSFormat	
netErrDNSServerFailure	
netErrDNSNonexistantName	
netErrDNSNIY	
netErrDNSRefused	

## Net Library

### Net Library Functions

---

netErrDNSImpossible  
netErrDNSNoRRS  
netErrDNSAborted  
netErrDNSBadProtocol  
netErrDNSTruncated  
netErrDNSNoRecursion  
netErrDNSIrrelevant  
netErrDNSNotInLocalCache  
netErrDNSNoPort

#### Sockets Equivalent

None

**Comments** This call looks up the name(s) of host(s) to use for sending an e-mail. The caller passes the name of the mail exchange in `mailNameP` and gets back a list of host names to which the mail message can be sent.

**See Also** [NetLibGetHostByAddr](#), [NetLibGetHostName](#)

## NetLibGetServByName

**Purpose** Looks up the port number for a standard TCP/IP service, given the desired protocol.

**Declared In** NetMgr.h

**Prototype** `NetServInfoPtr NetLibGetServByName  
(UInt16 libRefNum, const Char *servNameP,  
const Char *protoNameP, NetServInfoBufPtr bufP,  
Int32 timeout, Err *errP)`

**Parameters** -> `libRefNum` Reference number of the net library.

-> servNameP	Name of the service to look up. Possible services are "echo", "discard", "daytime", "qotd", "chargen", "ftp-data", "ftp", "telnet", "smtp", "time", "name", "finger", "pop2", "pop3", "nntp", "imap2".
-> protoNameP	Desired protocol to use, either "udp" or "tcp".
<- bufP	Pointer to a <a href="#">NetServInfoBufType</a> struct in which to store the results of the lookup.
-> timeout	Maximum timeout in system ticks; -1 means wait forever.
<- errP	Contains an error code if the return value is 0.

**Result** Returns a pointer to the [NetServInfoType](#) portion of bufP that contains results of the lookup. If the return value is 0, and error has occurred and errP contains one of the following values:

0	No error
netErrTimeout	Call timed out.
netErrNotOpen	The referenced net library has not been opened yet.
netErrUnknownProtocol	
netErrUnknownService	

**Sockets Equivalent** `struct servent *getservbyname (char *addr, char *proto);`

**Comments** This call is a convenience call for looking up a standard port number given the name of a service and the protocol to use.

**See Also** [NetLibGetHostByName](#)

## NetLibIFAttach

<b>Purpose</b>	Attach a new network interface.
<b>Declared In</b>	NetMgr.h
<b>Prototype</b>	Err NetLibIFAttach (UInt16 libRefNum, UInt32 ifCreator, UInt16 ifInstance, Int32 timeout)
<b>Parameters</b>	-> libRefNum      Reference number of the net library. -> ifCreator       Creator of interface to attach. -> ifInstance      Instance number of interface to attach. The instance number is one of the values returned by <a href="#">NetLibIFGet</a> . -> timeout          Timeout in ticks; -1 means infinite timeout.
<b>Result</b>	Returns one of the following values: 0                   Success. netErrInterfaceNotFound netErrTooManyInterfaces
<b>Sockets Equivalent</b>	None
<b>Comments</b>	This call can be used to attach a new network interface to the net library. Network interfaces are self-contained databases of type 'neti'. The ifCreator parameter to this function is used to locate the network interface database of the given creator.  If the net library is already open when this call is made, the network interface's database will be located and then called to initialize itself and attach itself to the protocol stack in real time. If the net library is not open when this call is made, the creator and instance number of the interface are stored in the active configuration. You need to save the active configuration using <a href="#">NetLibConfigSaveAs</a> if you want

the interface to be initialized and attached to the stack the next time the net library is opened.

**See Also** [NetLibIFGet](#), [NetLibIFDetach](#)

## NetLibIFDetach

**Purpose** Detach a network interface from the protocol stack.

**Declared In** NetMgr.h

**Prototype** Err NetLibIFDetach (UInt16 libRefNum,  
                  UInt32 ifCreator, UInt16 ifInstance,  
                  Int32 timeout)

**Parameters**

-> libRefNum	Reference number of the net library.
-> ifCreator	Creator of interface to detach.
-> ifInstance	Instance number of interface to detach.
-> timeout	Timeout in ticks; -1 means infinite timeout.

**Result** Returns one of the following values:

0 Success.

netErrInterfaceNotFound

**Sockets  
Equivalent** None

**Comments** If the net library is already open when this call is made, the interface is brought down and detached from the protocol stack in real time. If the net library is not open when this call is made, the creator and instance number of the interface are removed from the active configuration. You need to save the active configuration using [NetLibConfigSaveAs](#) if you don't want the interface to be attached the next time the library is opened.

**See Also** [NetLibIFGet](#), [NetLibIFAttach](#)

## NetLibIFDown

<b>Purpose</b>	Bring an interface down and hang up a connection.								
<b>Declared In</b>	NetMgr.h								
<b>Prototype</b>	<pre>Err NetLibIFDown (UInt16 libRefNum,                   UInt32 ifCreator, UInt16 ifInstance,                   Int32 timeout)</pre>								
<b>Parameters</b>	<table><tr><td>-&gt; libRefNum</td><td>Reference number of the net library.</td></tr><tr><td>-&gt; ifCreator</td><td>Creator of interface to attach.</td></tr><tr><td>-&gt; ifInstance</td><td>Instance number of interface to attach.</td></tr><tr><td>-&gt; timeout</td><td>Timeout in ticks; -1 means wait forever.</td></tr></table>	-> libRefNum	Reference number of the net library.	-> ifCreator	Creator of interface to attach.	-> ifInstance	Instance number of interface to attach.	-> timeout	Timeout in ticks; -1 means wait forever.
-> libRefNum	Reference number of the net library.								
-> ifCreator	Creator of interface to attach.								
-> ifInstance	Instance number of interface to attach.								
-> timeout	Timeout in ticks; -1 means wait forever.								
<b>Result</b>	Returns one of the following values: 0 Success. netErrNotOpen The referenced net library has not been opened yet. netErrInterfaceNotFound								
<b>Sockets Equivalent</b>	None								
<b>Comments</b>	The net library must be open before this call can be made. For dial-up interfaces, this call terminates a connection and hangs up the modem if necessary. <a href="#">NetLibClose</a> automatically brings down any attached interfaces, so this routine doesn't normally have to be called. If the interface is already down, this routine returns immediately with no error.								
<b>See Also</b>	<a href="#">NetLibIFGet</a> , <a href="#">NetLibIFAttach</a> , <a href="#">NetLibIFDetach</a> , <a href="#">NetLibIFUp</a>								

## NetLibIFGet

<b>Purpose</b>	Get the creator and instance number of an installed interface by index.								
<b>Declared In</b>	NetMgr.h								
<b>Prototype</b>	<pre>Err NetLibIFGet (UInt16 libRefNum, UInt16 index, UInt32 *ifCreatorP, UInt16 *ifInstanceP)</pre>								
<b>Parameters</b>	<table><tr><td>-&gt; libRefNum</td><td>Reference number of the net library.</td></tr><tr><td>-&gt; index</td><td>Index of the interface to get. Indices start at 0.</td></tr><tr><td>&lt;- ifCreatorP</td><td>The interface's creator.</td></tr><tr><td>&lt;- ifInstanceP</td><td>The interface's instance number.</td></tr></table>	-> libRefNum	Reference number of the net library.	-> index	Index of the interface to get. Indices start at 0.	<- ifCreatorP	The interface's creator.	<- ifInstanceP	The interface's instance number.
-> libRefNum	Reference number of the net library.								
-> index	Index of the interface to get. Indices start at 0.								
<- ifCreatorP	The interface's creator.								
<- ifInstanceP	The interface's instance number.								
<b>Result</b>	Returns one of the following values: <table><tr><td>0</td><td>Success.</td></tr><tr><td>netErrInvalidInterface</td><td>Index too high</td></tr><tr><td>netErrPrefNotFound</td><td>No current value for setting.</td></tr></table>	0	Success.	netErrInvalidInterface	Index too high	netErrPrefNotFound	No current value for setting.		
0	Success.								
netErrInvalidInterface	Index too high								
netErrPrefNotFound	No current value for setting.								
<b>Sockets Equivalent</b>	None								
<b>Comments</b>	To get a list of all installed interfaces, call this function with successively increasing indices starting from 0 until the error <code>netErrInvalidInterface</code> is returned. The <code>ifCreator</code> and <code>ifInstance</code> values returned from this call can then be used with the <a href="#">NetLibSettingGet</a> call to get more information about that particular interface.								
<b>See Also</b>	<a href="#">NetLibIFAttach</a> , <a href="#">NetLibIFDetach</a> , “ <a href="#">Settings for Interface Selection</a> ” in the <i>Palm OS Programmer’s Companion</i> , vol. II, <i>Communications</i>								

## NetLibIFSettingGet

<b>Purpose</b>	Retrieves a network interface specific setting.															
<b>Declared In</b>	NetMgr.h															
<b>Prototype</b>	<pre>Err NetLibIFSettingGet (UInt16 libRefNum, UInt32 ifCreator, UInt16 ifInstance, UInt16 setting, void *valueP, UInt16 *valueLenP)</pre>															
<b>Parameters</b>	<table><tr><td>-&gt; libRefNum</td><td>Reference number of the net library.</td></tr><tr><td>-&gt; ifCreator</td><td>Creator of the network interface.</td></tr><tr><td>-&gt; ifInstance</td><td>Instance number of the network interface.</td></tr><tr><td>-&gt; setting</td><td>Setting to retrieve; one of the NetIFSettingEnum constants.</td></tr><tr><td>&lt;- valueP</td><td>Space for return value of setting.</td></tr><tr><td>&lt;-&gt; valueLenP</td><td>On entry, size of valueP. On exit, actual size of setting.</td></tr></table>		-> libRefNum	Reference number of the net library.	-> ifCreator	Creator of the network interface.	-> ifInstance	Instance number of the network interface.	-> setting	Setting to retrieve; one of the NetIFSettingEnum constants.	<- valueP	Space for return value of setting.	<-> valueLenP	On entry, size of valueP. On exit, actual size of setting.		
-> libRefNum	Reference number of the net library.															
-> ifCreator	Creator of the network interface.															
-> ifInstance	Instance number of the network interface.															
-> setting	Setting to retrieve; one of the NetIFSettingEnum constants.															
<- valueP	Space for return value of setting.															
<-> valueLenP	On entry, size of valueP. On exit, actual size of setting.															
<b>Result</b>	<p>Returns one of the following values:</p> <table><tr><td>0</td><td>Success.</td></tr><tr><td>netErrUnknownSetting</td><td>Invalid setting constant.</td></tr><tr><td>netErrPrefNotFound</td><td>No current value for setting.</td></tr><tr><td>netErrBufTooSmall</td><td>valueP was too small to hold entire setting. Setting value was truncated to fit in valueP.</td></tr><tr><td>netErrUnimplemented</td><td></td></tr><tr><td>netErrInterfaceNotFound</td><td></td></tr><tr><td>netErrBufWrongSize</td><td></td></tr></table>		0	Success.	netErrUnknownSetting	Invalid setting constant.	netErrPrefNotFound	No current value for setting.	netErrBufTooSmall	valueP was too small to hold entire setting. Setting value was truncated to fit in valueP.	netErrUnimplemented		netErrInterfaceNotFound		netErrBufWrongSize	
0	Success.															
netErrUnknownSetting	Invalid setting constant.															
netErrPrefNotFound	No current value for setting.															
netErrBufTooSmall	valueP was too small to hold entire setting. Setting value was truncated to fit in valueP.															
netErrUnimplemented																
netErrInterfaceNotFound																
netErrBufWrongSize																
<b>Sockets Equivalent</b>	None															

**Comments**

This call can be used to retrieve the current value of any network interface setting. The caller must pass a pointer to a buffer to hold the return value (`valueP`), the size of the buffer (`*valueLenP`), and the setting ID (`setting`). The setting ID is one of the constants in the `NetIFSettingEnum` type.

Some settings, such as the login script, are variable size. For these types of settings, you can obtain the actual size required for the buffer by passing 0 for `*valueLenP`. The required size is returned in `valueLenP`.

[Table 61.2](#) lists the network interface settings and the size of each setting. Some are only applicable to certain types of interfaces. Settings not applicable to a specific interface can be safely ignored and not set to any particular value.

**Table 61.2 Network Interface Settings**

<b>netIFSetting...</b>	<b>Type</b>	<b>Description</b>
ResetAll	void	Use with <a href="#">NetLibIFSettingSet</a> only. This clears all other settings for the interface to their default values.
Up	UInt8	Read-only. true if interface is currently up.
Name	Char [32]	Read-only. Name of this interface.
ReqIPAddr	UInt32	IP address of interface.
SubnetMask	UInt32	Subnet mask for interface. Doesn't need to be specified for PPP or SLIP type connections.
Broadcast	UInt32	Broadcast address for interface. Doesn't need to be specified for PPP or SLIP type connections.
Username	Char [32]	User name. Only required if the login script uses the user name substitution escape sequence in it. Call <a href="#">NetLibIFSettingSet</a> with a valueLen of 0 to remove this setting.
Password	Char [32]	Password. Only required if the login script uses the password substitution escape sequence in it. Call <a href="#">NetLibIFSettingSet</a> with a valueLen of 0 to remove this setting. If the login script uses password substitution and no password setting is set, the user will be prompted for a password at connect time.
AuthUsername	Char [32]	Authentication user name. Only required if the authentication protocol uses a different user name than the what's in the netIFSettingUsername setting. If this setting is empty (valueLen of 0), the Username setting will be used instead. Call <a href="#">NetLibIFSettingSet</a> with a valueLen of 0 to remove this setting.

**Table 61.2 Network Interface Settings (*continued*)**

<b>netIFSetting...</b>	<b>Type</b>	<b>Description</b>
AuthPassword	Char [32]	Authentication password. If "\$" then the user will be prompted for the authentication password at connect time. Else, if 0 length, then the netIFSettingPassword setting or the result of its prompt will be used instead. Call <a href="#">NetLibIFSettingSet</a> with a valueLen of 0 to remove this setting.
ServiceName	Char []	Service name. Used for display purposes while showing the connection progress dialog box. Call <a href="#">NetLibIFSettingSet</a> with a valueLen of 0 to remove this setting.
LoginScript	Char []	Login script. Only required if the particular service requires a login sequence. Call <a href="#">NetLibIFSettingSet</a> with a valueLen of 0 to remove this setting. See below for a description of the login script format.
ConnectLog	Char []	Connect log. Generally, this setting is just retrieved, not set. It contains a log of events from the most recent login. To clear this setting, call <a href="#">NetLibIFSettingSet</a> with a valueLen of 0.
InactivityTimeout	UInt16	Maximum number of seconds of inactivity allowed. Set to 0 to ignore.
EstablishmentTimeout	UInt16	Maximum delay, in seconds, allowed between each stage of connection establishment or login script line. Must be non-zero.
DynamicIP	UInt8	If non-zero, negotiate for an IP address. If zero, the IP address specified in the netIFSettingReqIPAddr setting will be used. Default is false.
VJCompEnable	UInt8	If non-zero, enable VJ header compression. Default is true for PPP, false for SLIP, and true for CSLIP.

**Table 61.2 Network Interface Settings (*continued*)**

<b>netIFSetting...</b>	<b>Type</b>	<b>Description</b>
VJCompSlots	UInt8	Number of slots to use for VJ compression. Default is 4 for PPP and 16 for SLIP and CSLIP. More slots require more memory so it is best to keep this number to a minimum.
MTU	UInt16	Maximum transmission unit in octets. Currently not implemented in SLIP or PPP.
AsyncCtlMap	UInt32	Bit mask of characters to escape for PPP. Default is 0.
PortNum	UInt16	Which serial communication port to use. Port 0 is the only port available on the device.
BaudRate	UInt32	Serial port baud rate to use in bits per second.
FlowControl	UInt8	If bit 0 is 1, use hardware handshaking on the serial port. Default is no hardware handshaking.
StopBits	UInt8	Number of stop bits. Default is 1.
ParityOn	UInt8	true if parity detection enabled. Default is false.
ParityEven	UInt8	true for even parity detection. Default is true.
UseModem	UInt8	If true, dial-up through modem. If false, go direct over serial port
PulseDial	UInt8	If true, pulse dial modem. Else, tone dial. Default is tone dial.
ModemInit	Char []	Zero-terminated modem initialization string, not including the "AT". If not specified (valueLen of 0), the modem initialization string from system preferences are used.
ModemPhone	Char []	Zero-terminated modem phone number string. Only required if netIFSettingUseModem is true.

**Table 61.2 Network Interface Settings (*continued*)**

<b>netIFSetting...</b>	<b>Type</b>	<b>Description</b>
RedialCount	UInt16	Number of times to re-dial modem when trying to establish a connection. Only required if netIFSettingUseModem is true.
DNSQuery	UInt8	true if PPP queries for DNS address. The default is true.
TraceBits	UInt32	A bitfield of various trace bits. See " <a href="#">Tracing Bits</a> ." An application can get a list of events in the trace buffer using the <a href="#">NetLibMaster</a> call. Each interface has its own trace bits setting so that trace event recording in each interface can be selectively enabled or disabled.
ActualIPAddr	UInt32	Read-only. The actual IP address that the interface ends up using. The login script execution engine stores the result of the "g" (get IP address) command here as does the PPP negotiation logic.
ServerIPAddr	UInt32	Read-only. The IP address of the PPP server we're connected to.
BringDownOnPowerDown	UInt8	true if the interface is brought down when the Palm OS device is turned off.
RawMode	UInt32	Specifies if the interface is in raw mode. The net library places an interface in raw mode when it is bound to a raw socket in the raw domain. Raw sockets are available in Palm OS version 3.0 and higher.
DriverVersion	Char[20]	Read-only. The version number of the network interface device driver. This setting is defined only if <a href="#">5.1 New Feature Set</a> is present.
FirmwareVersion	Char[20]	Read-only. The firmware version of the network interface device, if any. This setting is defined only if <a href="#">5.1 New Feature Set</a> is present.

**Table 61.2 Network Interface Settings (*continued*)**

<b>netIFSetting...</b>	<b>Type</b>	<b>Description</b>
FirmwareDate	UInt32	Read-only. Firmware date in seconds since midnight, January 1, 1904. This setting is defined only if <a href="#">5.1 New Feature Set</a> is present.
80211Device	UInt8	Read-only. Indicates whether or not the interface supports IEEE 802.11 wireless networking. This setting is defined only if <a href="#">5.1 New Feature Set</a> is present.
80211ESSID	Char[32]	For IEEE 802.11 interfaces only. The ESS ID of the radio. This setting is defined only if <a href="#">5.1 New Feature Set</a> is present.
80211AccessPointBSSI D	UInt8[6]	Read-only. For IEEE 802.11 interfaces only. The BSS ID (MAC address) of the access point to which the radio is connected. This setting is defined only if <a href="#">5.1 New Feature Set</a> is present.
80211AssociationStatus	UInt8	Read-only. For IEEE 802.11 interfaces only. true if the radio is associated. This setting is defined only if <a href="#">5.1 New Feature Set</a> is present.
80211MKKCallSign	Char[15]	Read-only. For IEEE 802.11 interfaces with radios programmed for operation in Japan only. The MKK call sign. This setting is defined only if <a href="#">5.1 New Feature Set</a> is present.
80211CountryText	Char[34]	Read-only. For IEEE 802.11 interfaces only. The radio's country code, which the radio uses to check if it operates within a particular country's regulations. This setting is defined only if <a href="#">5.1 New Feature Set</a> is present.

**See Also** [NetLibIFSettingSet](#), [NetLibSettingGet](#), [NetLibSettingSet](#), “[Interface Specific Settings](#)” in the *Palm OS Programmer’s Companion*, vol. II, *Communications*

## NetLibIFSettingSet

<b>Purpose</b>	Sets a network interface specific setting.												
<b>Declared In</b>	NetMgr.h												
<b>Prototype</b>	<pre>Err NetLibIFSettingSet (UInt16 libRefNum,                         UInt32 ifCreator, UInt16 ifInstance,                         UInt16 setting, void *valueP, UInt16 valueLen)</pre>												
<b>Parameters</b>	<table><tr><td>-&gt; libRefNum</td><td>Reference number of the net library.</td></tr><tr><td>-&gt; ifCreator</td><td>Creator of the network interface.</td></tr><tr><td>-&gt; ifInstance</td><td>Instance number of the network interface.</td></tr><tr><td>-&gt; setting</td><td>The setting to set, one of the NetIFSettingEnum constants. See <a href="#">Table 61.2</a>.</td></tr><tr><td>-&gt; valueP</td><td>Space new value of setting.</td></tr><tr><td>-&gt; valueLen</td><td>Size of new setting.</td></tr></table>	-> libRefNum	Reference number of the net library.	-> ifCreator	Creator of the network interface.	-> ifInstance	Instance number of the network interface.	-> setting	The setting to set, one of the NetIFSettingEnum constants. See <a href="#">Table 61.2</a> .	-> valueP	Space new value of setting.	-> valueLen	Size of new setting.
-> libRefNum	Reference number of the net library.												
-> ifCreator	Creator of the network interface.												
-> ifInstance	Instance number of the network interface.												
-> setting	The setting to set, one of the NetIFSettingEnum constants. See <a href="#">Table 61.2</a> .												
-> valueP	Space new value of setting.												
-> valueLen	Size of new setting.												
<b>Result</b>	Returns one of the following values:  0 Success.  netErrUnknownSetting Invalid setting constant.  netErrPrefNotFound No current value for setting.  netErrUnimplemented  netErrInterfaceNotFound  netErrBufWrongSize  netErrReadOnlySetting												
<b>Sockets Equivalent</b>	None												
<b>Comments</b>	This call can be used to set the current value of any network interface setting. The caller must pass a pointer to a buffer which												

holds the new value (valueP), the size of the buffer (valueLen), and the setting ID (setting).

See [NetLibIFSettingGet](#) for an explanation of each of the settings.

Of particular interest is the netIFSettingResetAll setting, which, if used, resets all settings for the interface to their default values. When using this setting, valueP and valueLen are ignored.

**See Also** [NetLibIFSettingGet](#), [NetLibSettingGet](#),  
[NetLibSettingSet](#), “[Interface Specific Settings](#)” in the *Palm OS Programmer’s Companion*, vol. II, *Communications*

## NetLibIFUp

**Purpose** Bring an interface up and establish a connection.

**Declared In** NetMgr.h

**Prototype** Err NetLibIFUp (UInt16 libRefNum,  
                  UInt32 ifCreator, UInt16 ifInstance)

**Parameters** -> libRefNum      Reference number of the net library.  
                -> ifCreator      Creator of interface to attach.  
                -> ifInstance      Instance number of interface to attach.

**Result** Returns one of the following values:

- |                         |   |
|-------------------------|---|
| 0                       | Success.  |
| netErrNotOpen           | The referenced net library has not been opened yet. |
| netErrInterfaceNotFound |   |
| netErrUserCancel        |   |
| netErrBadScript         |   |
| netErrPPPTimeout        |   |

	netErrAuthFailure netErrPPPAddressRefused
<b>Sockets Equivalent</b>	None
<b>Comments</b>	The net library must be open before this call can be made. For dial-up interfaces, this call will dial up the modem if necessary and run through the connect script to establish the connection.  If the interface is already up, this routine returns immediately with no error. This call doesn't take a timeout parameter because it relies on each interface to have its own established timeout setting.
<b>See Also</b>	<a href="#">NetLibIFGet</a> , <a href="#">NetLibIFAttach</a> , <a href="#">NetLibIFDetach</a> , <a href="#">NetLibIFDown</a>

## NetLibMaster

<b>Purpose</b>	Retrieves the network statistics, interface statistics, and the contents of the trace buffer.	
<b>Declared In</b>	NetMgr.h	
<b>Prototype</b>	Err NetLibMaster (UInt16 libRefNum, UInt16 cmd, NetMasterPBPtr pbP, Int32 timeout)	
<b>Parameters</b>	-> libRefNum	Reference number of the net library.
	-> cmd	Function to perform (NetMasterEnum type). The following commands are supported:  <a href="#">netMasterInterfaceInfo</a> <a href="#">netMasterInterfaceStats</a> <a href="#">netMasterIPStats</a> <a href="#">netMasterICMPStats</a> <a href="#">netMasterUDPStats</a> <a href="#">netMasterTCPStats</a> <a href="#">netMasterTraceEventGet</a>
	<-> pbP	Command parameter block.

## Net Library

### Net Library Functions

---

-> timeout      Timeout in ticks; -1 means wait forever.

**Result**    Returns one of the following values:

0                  No error

netErrNotOpen    The referenced net library has not been opened yet.

netErrParamErr

netErrUnimplemented

#### Sockets Equivalent

None

#### Comments

This call allows applications to get detailed information about the net library. This information is usually helpful in debugging network configuration problems.

This function takes a command word (cmd) and parameter block pointer (pbP) as arguments and returns its results in the parameter block on exit. Which values you must specify in the parameter block and which values are returned are specific to the command you specify.

### netMasterInterfaceInfo

The pbP->interfaceInfo struct specifies interface information.

-> index                         Index of interface to fetch info about.

<- creator                         Creator of interface.

<- instance                         Instance of interface.

<- netIFP                             Private interface info pointer.

<- drvrName                         Driver type that interface uses ("PPP", "SLIP", etc.).

<- hwName                             Hardware driver name ("Serial Library", etc.).

<- localNetHdrLen	Number of bytes in local net header.
<- localNetTrailerLen	Number of bytes in local net trailer.
<- localNetMaxFrame	Local net maximum frame size.
<- ifName	Interface name with instance number concatenated.
<- driverUp	true if interface driver is up.
<- ifUp	true if interface media layer is up.
<- hwAddrLen	Length of interface's hardware address.
<- hwAddr	Interface's hardware address.
<- mtu	Maximum transfer unit of interface.
<- speed	Speed in bits per second.
<- lastStateChange	Time in milliseconds of last state change.
<- ipAddr	IP address of interface.
<- subnetMask	Subnet mask of local network.
<- broadcast	Broadcast address of local network.

### **netMasterInterfaceStats**

The pbP->interfaceStats structure specifies interface statistics.

-> index	Index of interface to fetch info about.
<- inOctets	Number of octets received.
<- inUcastPkts	Number of packets received.
<- inNUcastPkts	Number of broadcast packets received.
<- inDiscards	Number of incoming packets that were discarded.

<- inErrors	Number of packet errors encountered.
<- inUnknownProtos	Number of unknown protocols encountered.
<- outOctets	Number octets sent.
<- outUcastPkts	Number of packets sent.
<- outNUcastPkts	Number of broadcast packets sent.
<- outDiscards	Number of packets discarded.
<- outErrors	Number of outbound packet errors.

### **netMasterIPStats**

The pbP->ipStats structure contains statistics about the IP protocol. See NetMgr.h for a complete list of statistics returned.

### **netMasterICMPStats**

The pbP->icmpStats structure contains statistics about the ICMP protocol. See NetMgr.h for a complete list of statistics returned.

### **netMasterUDPStats**

The pbP->udpStats structure contains statistics about the UDP protocol. See NetMgr.h for a complete list of statistics returned.

### **netMasterTCPStats**

The pbP->tcpStats structure contains statistics about the TCP protocol. See NetMgr.h for a complete list of statistics returned.

### **netMasterTraceEventGet**

The pbP->traceEventGet structure contains a trace event.

- > index Index of event to fetch.
- <- textP Pointer to text string to return event in. Should be at least 256 bytes long.

**See Also** [NetLibSettingSet](#)

## NetLibOpen

**Purpose** Opens and initializes the net library.

**Declared In** NetMgr.h

**Prototype** Err NetLibOpen (UInt16 libRefnum,  
                  UInt16 \*netIFErrsP)

**Parameters** -> libRefnum      Reference number of the net library.  
                  -< netIFErrsP      First error encountered when bringing up  
  network interfaces. (See [NetLibIFUp](#) for a list  
  of possible values.)

**Result** Returns one of the following values:

0                        No error.

netErrAlreadyOpen

Not really an error; returned if library was  
already open and the open count was simply  
incremented.

netErrOutOfMemory

Not enough memory available to open the  
library.

netErrNoInterfaces

Incorrect setup.

netErrPrefNotFound

Incorrect setup.

**Comments** Applications must call this function before using the net library. If the net library was already open, NetLibOpen increments its open count. Otherwise, it opens the library, initializes it, starts up the net protocol stack component of the library as a separate task, and brings up all attached network interfaces.

NetLibOpen uses settings saved in the net library's preferences database during initialization. These settings include the interfaces to attach, the IP addresses, etc. It's assumed that these settings have

been previously set up by a preference panel or equivalent so an application doesn't normally have to set them up before calling NetLibOpen.

If any of the attached interfaces fails to come up, \*netIFErrsP will contain the error number of the first interface that encountered a problem.

**Compatibility** NetLibOpen behaves slightly differently in version 3.2 and later than it does in previous releases. In version 3.2 and later, NetLibOpen calls [NetLibOpenConfig](#) specifying the default configuration. NetLibOpenConfig reverts all settings to their saved, default values before opening the net library.

**See Also** [SysLibFind](#), [NetLibClose](#), [NetLibOpenCount](#)



## NetLibOpenConfig

**Purpose** Opens and initializes the net library with the specified configuration.

**Prototype** Err NetLibOpenConfig (UInt16 refNum,  
                  UInt16 configIndex, UInt32 openFlags,  
                  UInt16 \*netIFErrP)

**Parameters**

-> refNum	Reference number of the net library.
-> configIndex	Index of the configuration to use. 0 means use the default configuration as defined in the Network preferences panel.
-> openFlags	Not used. Pass 0 for this parameter.
<- netIFErrP	Pointer to return error code for interfaces.

**Result** Returns one of the following values:  
0                   No error.  
memErrNotEnoughSpace

`netErrAlreadyOpen`

Not really an error; returned if library was already open and the open count was simply incremented.

`netErrAlreadyOpenWithAnotherConfig`

Another application has the net library open with a configuration that is incompatible with the one specified.

`netErrBufTooSmall`

`netErrConfigAliasErr`

A configuration alias was specified, but the alias could not be resolved.

`netErrConfigCantDelete`

`netErrConfigEmpty`

The configuration is not defined.

`netErrConfigNotFound`

The specified configuration index is invalid.

`netErrInterfaceNotFound`

`netErrOutOfCmdBlocks`

`netErrOutOfMemory`

Not enough memory available to open the library.

`netErrNoInterfaces`

Incorrect setup.

`netErrParamErr`

`netErrPrefNotFound`

Incorrect setup.

`netErrTimeout`

**Sockets  
Equivalent** None

**Comments** Use this routine instead of [NetLibOpen](#) when you want to open the net library with a non-default configuration. If the default net library configuration is not suitable for your application, you may

## Net Library

### Net Library Functions

---

use one of the built-in aliases to specify a configuration that is suitable (see “[Configuration Aliases](#)”).

`NetLibOpenConfig` tries to open the net library and initialize it with the specified configuration. If another application has the net library open with an incompatible configuration, it returns an error. If the net library is in the close-wait state, this function completely closes the net library and then reopens it using the new configuration. If the net library can be opened with the new configuration, `NetLibOpenConfig` first saves the current configuration so that it can be restored when your application closes the net library.

Typically, applications use the [`NetLibConfigList`](#) function to obtain the list of available configurations and present this list to the user. Then they call [`NetLibConfigIndexFromName`](#) with the user’s selection to get the index of the configuration that the user selected.

The constant `netConfigIndexCurSettings` specifies the current configuration. You can specify `netConfigIndexCurSettings` as the `configIndex` for testing purposes.

**Compatibility** Implemented only if [`3.2 New Feature Set`](#) is present.

**See Also** [`NetLibOpen`](#), [`SysLibFind`](#), [`NetLibClose`](#), [`NetLibOpenCount`](#)

## NetLibOpenCount

**Purpose** Retrieves the open count of the net library.

**Declared In** `NetMgr.h`

**Prototype** `Err NetLibOpenCount (UInt16 refNum,  
                  UInt16 *countP)`

**Parameters** `-> refNum` Reference number of the net library.

<- countP Contains the open count of the net library upon return.

**Result** Always returns 0.

**Sockets  
Equivalent** None.

**Comments** This call will most likely only be used by the Network preferences panel. Most applications will simply call [NetLibOpen](#) unconditionally during startup and [NetLibClose](#) when they exit.

## NetLibReceive

**Purpose** Receive data from a socket into a single buffer.

**Declared In** NetMgr.h

**Prototype** Int16 NetLibReceive (UInt16 libRefNum,  
NetSocketRef socket, void \*bufP, UInt16 bufLen,  
UInt16 flags, void \*fromAddrP, UInt16 \*fromLenP,  
Int32 timeout, Err \*errP)

<b>Parameters</b>	-> libRefNum	Reference number of the net library.
	-> socket	Descriptor for the open socket.
	<- bufP	Pointer to buffer to hold received data.
	-> bufLen	Length of bufP buffer.
	-> flags	One or more netIOFlagxxx flags. See " <a href="#">I/O Flags</a> ."
	<- fromAddrP	Pointer to buffer to hold address of sender (a <a href="#">NetSocketAddress</a> ).
	<-> fromLenP	On entry, size of fromAddrP buffer. On exit, actual size of returned address in fromAddrP.
	-> timeout	Maximum timeout in system ticks; -1 means wait forever.

## Net Library

### Net Library Functions

---

<- errP Contains an error code if the return value is -1.

**Result** Returns the number of bytes successfully received. If the return value is 0, the socket has been shut down by the remote host. If the return value is -1, an error has occurred, and errP contains one of the following values:

0 No error.

netErrTimeout Call timed out.

netErrNotOpen The referenced net library has not been opened yet.

netErrParamErr

netErrSocketNotOpen

netErrWouldBlock

netErrUserCancel

#### Sockets Equivalent

```
int recvfrom (int socket, const void *bufP,  
int bufLen, int flags, const void *fromAddrP,  
int *fromLenP);  
  
int recv (int socket, const void *bufP,  
int bufLen, int flags);  
  
int read (int socket, const void *bufP,  
int bufLen);
```

#### Comments

For stream-based sockets, this call reads whatever bytes are available and returns the number of bytes actually read into the caller's buffer. If there is no data available, this call will block until at least one byte arrives, until the socket is shut down by the remote host, or until a timeout occurs.

For datagram-based sockets, this call reads a complete datagram and returns the number of bytes in the datagram. If the caller's buffer is not large enough to hold the entire datagram, the end of the datagram is discarded. If a datagram is not available, this call will block until one arrives, or until the call times out.

The data is read into a single buffer pointed to by bufP.

**See Also** [NetLibReceive](#), [NetLibDmReceive](#), [NetUReadN](#), [NetLibSend](#), [NetLibSendPB](#)

## NetLibReceivePB

**Purpose** Receive data from a socket into a multi-buffer gather-read array.

**Declared In** NetMgr.h

**Prototype** `Int16 NetLibReceivePB (UInt16 libRefNum,  
NetSocketRef socket, NetIOParamType *pbP,  
UInt16 flags, Int32 timeout, Err *errP)`

**Parameters**

-> libRefNum	Reference number of the net library.
-> socket	Descriptor for the open socket.
-> pbP	Pointer to parameter block containing buffer info.
-> flags	One or more netIOFlagxxx flags. See “ <a href="#">I/O Flags</a> .”
-> timeout	Maximum timeout in system ticks; -1 means wait forever.
<- errP	Contains an error code if the return value is -1.

**Result** Returns the number of bytes successfully received. Returns 0 if the socket has been shut down by the remote host. If the return value is -1, an error has occurred, and errP contains one of the following values:

0	No error.
netErrTimeout	Call timed out.
netErrNotOpen	The referenced net library has not been opened yet.
netErrParamErr	
netErrSocketNotOpen	

## Net Library

### Net Library Functions

---

netErrWouldBlock

**Sockets Equivalent** int recvmsg (int socket, const struct msghdr \*pbP,  
int flags);

**Comments** The pbP parameter is a pointer to a NetIOPParamType structure. NetIOPParamType is defined as follows:

```
typedef struct {
    UInt8      *addrP;
    UInt16     addrLen;
    NetIOVecPtr iov;
    UInt16     iovLen;
    UInt8      *accessRights;
    UInt16     accessRightsLen;
} NetIOPParamType, *NetIOPParamPtr;
```

You provide the following information in this struct:

addrP	Address of sender, set by NetLibReceivePB. Set to 0 if you don't require this field.
addrLen	Length of *addrP.
iov	Array of buffers into which the data should be received. NetIOVecPtr is a pointer to a NetIOVecType structure, which has two fields:  bufP              Pointer to a buffer. bufLen            Length of bufP.
iovLen	Length of the iov array.
accessRights	Access rights. This field currently isn't used and should be set to 0.
accessRightsLen	Length of the *accessRights. This field currently isn't used and should be set to 0.

For stream-based sockets, this call reads whatever bytes are available and returns the number of bytes actually read into the caller's buffer. If no data is available, this call will block until at least

one byte arrives, until the socket is shut down by the remote host, or until a timeout occurs.

For datagram-based sockets, this call reads a complete datagram and returns the number of bytes in the datagram. If the caller's buffer is not large enough to hold the entire datagram, the end of the datagram is discarded. If a datagram is not available, this call will block until one arrives, or until the call times out.

The data is read into the gather-read array specified by the pbP->iov array.

**See Also** [NetLibReceive](#), [NetLibDmReceive](#), [NetLibSend](#),  
[NetLibSendPB](#)

## NetLibSelect

**Purpose** Blocks until I/O is ready on one or more descriptors, where a descriptor can represent socket input, socket output, or a user input event like a pen tap or key press.

**Declared In** NetMgr.h

**Prototype**

```
Int16 NetLibSelect (UInt16 libRefNum,  
                    UInt16 width, NetFDSetType *readFDs,  
                    NetFDSetType *writeFDs, NetFDSetType *exceptFDs,  
                    Int32 timeout, Err *errP)
```

**Parameters**

-> libRefNum	Reference number of the net library.
-> width	Number of descriptor bits to check in the readFDs, writeFDs, and exceptFDs descriptor sets.
<-> readFDs	Pointer to 32-bit NetFDSetType containing set of bits representing descriptors to check for input.
<-> writeFDs	Pointer to 32-bit NetFDSetType containing set of bits representing descriptors to check for output.

<-> exceptFDs      Pointer to 32-bit NetFDSetType containing set of bits representing descriptors to check for exception conditions. This parameter is ignored. Upon return, its bits are always cleared.

-> timeout      Maximum timeout in system ticks; -1 means wait forever.

<- errP      Contains an error code if the return value is -1.

**Result**      Returns the sum total number of ready file descriptors in \*readFDs, \*writeFDs, and \*exceptFDs. Returns 0 upon timeout. If the return value is -1, an error has occurred, and errP contains one of the following values:

0      No error

netErrTimeout      Call timed out.

netErrNotOpen      The referenced net library has not been opened yet.

**Sockets Equivalent**

```
int select (int width, fd_set *readfds,
fd_set *writefds, fd_set *exceptfds,
struct timeval *timeout);
```

**Comments**      This call blocks until one or more descriptors are ready for I/O. In the Palm OS environment, a descriptor is either a NetSocketRef or the "stdin" descriptor, sysFileDescStdIn. The sysFileDescStdIn descriptor will be ready for input whenever a user event is available like a pen tap or key press.

The caller should set which bits in each descriptor set need to be checked by using the netFDZero and netFDSet macros. After this call returns, the macro netFDIsSet can be used to determine which descriptors in each set are actually ready.

On exit, the total number of ready descriptors is returned and each descriptor set is updated with the appropriate bits set for each ready descriptor in that set.

The following example illustrates how to use this call to check for input on a socket or a user event:

```
Err      err;
NetSocketRef    socket;
NetFDSetType    readFDs,writeFDs,exceptFDs;
Int16        numFDs;
UIInt16       width;

// Create the descriptor sets
netFDZero(&readFDs);
netFDZero(&writeFDs);
netFDZero(&exceptFDs);
netFDSet(sysFileDescStdIn, &readFDs);
netFDSet(socket, &readFDs);

// Calculate the max descriptor number and
// use that +1 as the max width.
// Alternatively, we could simply use the
// constant netFDSetSize as the width which
// is simpler but makes the NetLibSelect call
// slightly slower.
width = sysFileDescStdIn;
if (socket > width) width = socket;

// Wait for any one of the descriptors to be
// ready.
numFDs = NetLibSelect(AppNetRefnum, width+1,
                      &readFDs, &writeFDs, &exceptFDs,
                      AppNetTimeout, &err);
```

Also see the `NetSample` example application in the `Palm OS Examples` folder. The function `CmdTelnet` in the file `CmdTelnet.c` shows how to use the Berkeley sockets `select` function and how to interpret the results.

**See Also** [NetLibSocketOptionSet](#)

## NetLibSend

<b>Purpose</b>	Send data to a socket from a single buffer.																		
<b>Declared In</b>	NetMgr.h																		
<b>Prototype</b>	<pre>Int16 NetLibSend (UInt16 libRefNum, NetSocketRef socket, void *bufP, UInt16 bufLen, UInt16 flags, void *toAddrP, UInt16 toLen, Int32 timeout, Err *errP)</pre>																		
<b>Parameters</b>	<table><tr><td>-&gt; libRefNum</td><td>Reference number of the net library.</td></tr><tr><td>-&gt; socket</td><td>Descriptor for the open socket.</td></tr><tr><td>-&gt; bufP</td><td>Pointer to data to write.</td></tr><tr><td>-&gt; bufLen</td><td>Length of data to write</td></tr><tr><td>-&gt; flags</td><td>One or more <code>netIOFlagxxx</code> flags. See “<a href="#">I/O Flags</a>.”</td></tr><tr><td>-&gt; toAddrP</td><td>Address to send to (a pointer to a <a href="#">NetSocketAddrType</a>), or 0.</td></tr><tr><td>-&gt; toLen</td><td>Size of <code>toAddrP</code> buffer.</td></tr><tr><td>-&gt; timeout</td><td>Maximum timeout in system ticks; -1 means wait forever.</td></tr><tr><td>&lt;- errP</td><td>Contains an error code if the return value is -1.</td></tr></table>	-> libRefNum	Reference number of the net library.	-> socket	Descriptor for the open socket.	-> bufP	Pointer to data to write.	-> bufLen	Length of data to write	-> flags	One or more <code>netIOFlagxxx</code> flags. See “ <a href="#">I/O Flags</a> .”	-> toAddrP	Address to send to (a pointer to a <a href="#">NetSocketAddrType</a> ), or 0.	-> toLen	Size of <code>toAddrP</code> buffer.	-> timeout	Maximum timeout in system ticks; -1 means wait forever.	<- errP	Contains an error code if the return value is -1.
-> libRefNum	Reference number of the net library.																		
-> socket	Descriptor for the open socket.																		
-> bufP	Pointer to data to write.																		
-> bufLen	Length of data to write																		
-> flags	One or more <code>netIOFlagxxx</code> flags. See “ <a href="#">I/O Flags</a> .”																		
-> toAddrP	Address to send to (a pointer to a <a href="#">NetSocketAddrType</a> ), or 0.																		
-> toLen	Size of <code>toAddrP</code> buffer.																		
-> timeout	Maximum timeout in system ticks; -1 means wait forever.																		
<- errP	Contains an error code if the return value is -1.																		
<b>Result</b>	Returns the number of bytes successfully sent. Returns 0 if the socket has been shut down by the remote host. If the return value is -1, an error has occurred, and <code>errP</code> contains one of the following values: <table><tr><td>0</td><td>No error.</td></tr><tr><td>netErrTimeout</td><td>Call timed out.</td></tr><tr><td>netErrNotOpen</td><td>The referenced net library has not been opened yet.</td></tr><tr><td>netErrParamErr</td><td></td></tr><tr><td>netErrSocketNotOpen</td><td></td></tr></table>	0	No error.	netErrTimeout	Call timed out.	netErrNotOpen	The referenced net library has not been opened yet.	netErrParamErr		netErrSocketNotOpen									
0	No error.																		
netErrTimeout	Call timed out.																		
netErrNotOpen	The referenced net library has not been opened yet.																		
netErrParamErr																			
netErrSocketNotOpen																			

```
netErrMessageTooBig  
netErrSocketNotConnected  
netErrSocketClosedByRemote  
netErrIPCantFragment  
netErrIPNoRoute  
netErrIPNoSrc  
netErrIPNoDst  
netErrIPktOverflow  
netErrOutOfCmdBlocks  
netErrOutOfPackets  
netErrInterfaceNotFound  
netErrInterfaceDown  
netErrUnreachableDest  
netErrNoMultiPktAddr  
netErrWouldBlock
```

**Sockets  
Equivalent**

```
int sendto (int socket, const void *bufP,  
int bufLen, int flags, const void *toAddrP,  
int toLen);  
  
int send (int socket, const void *bufP,  
int bufLen, int flags);  
  
int write (int socket, const void *bufP,  
int bufLen,);
```

**Comments**

This call attempts to write data to the specified socket and returns the number of bytes actually sent, which may be less than or equal to the requested number of bytes. The data is passed in a single buffer that bufP points to.

For datagram sockets, you must only send a single packet at a time. If the data is too large to fit in a single UDP packet (1536 bytes), no data is sent and -1 is returned.

The `toAddrP` field applies only to datagram sockets without an existing connection. An error is returned if the datagram socket was previously connected and `toAddrP` is specified. Stream-based sockets, by definition, must have a connection established with a remote host before data can be written. Raw sockets (supported in Palm OS version 3.0 and higher) must construct the entire IP header, including the destination address, before data can be sent; thus, the address is taken from the data to be sent.

If there isn't enough buffer space to send any data, this call will block until there is enough buffer space, or until a timeout.

---

**NOTE:** For stream-based sockets, this call may write only a portion of the desired data. It always returns the number of bytes actually written. Consequently, the caller should be prepared to call this routine repeatedly until the desired number of bytes have been written, or until it returns 0 or -1.

---

**See Also** [NetLibSendPB](#), [NetUWriteN](#), [NetLibReceive](#),  
[NetLibReceivePB](#), [NetLibDmReceive](#)

## NetLibSendPB

<b>Purpose</b>	Send data to a socket from a scatter-write array.						
<b>Declared In</b>	NetMgr.h						
<b>Prototype</b>	<pre>Int16 NetLibSendPB (UInt16 libRefNum, NetSocketRef socket, NetIOParamType *pbP, UInt16 flags, Int32 timeout, Err *errP)</pre>						
<b>Parameters</b>	<table><tr><td>-&gt; libRefNum</td><td>Reference number of the net library.</td></tr><tr><td>-&gt; socket</td><td>Descriptor for the open socket.</td></tr><tr><td>-&gt; pbP</td><td>Pointer to parameter block containing buffer info. See the description in <a href="#">NetLibReceivePB</a>.</td></tr></table>	-> libRefNum	Reference number of the net library.	-> socket	Descriptor for the open socket.	-> pbP	Pointer to parameter block containing buffer info. See the description in <a href="#">NetLibReceivePB</a> .
-> libRefNum	Reference number of the net library.						
-> socket	Descriptor for the open socket.						
-> pbP	Pointer to parameter block containing buffer info. See the description in <a href="#">NetLibReceivePB</a> .						

-> flags	One or more netIOFlagxxx flags. See “ <a href="#">I/O Flags</a> .”
-> timeout	Maximum timeout in system ticks; -1 means wait forever.
<- errP	Contains an error code if the return value is -1.

**Result** Returns the number of bytes successfully sent. Returns 0 if the socket has been shut down by the remote host. If the return value is -1, an error has occurred, and `errP` contains one of the following values:

0	No error.
netErrTimeout	Call timed out.
netErrNotOpen	The referenced net library has not been opened yet.
netErrParamErr	
netErrSocketNotOpen	
netErrMessageTooBig	
netErrSocketNotConnected	
netErrSocketClosedByRemote	
netErrIPCantFragment	
netErrIPNoRoute	
netErrIPNoSrc	
netErrIPNoDst	
netErrIPktOverflow	
netErrOutOfCmdBlocks	
netErrOutOfPackets	
netErrInterfaceNotFound	
netErrInterfaceDown	
netErrUnreachableDest	
netErrNoMultiPktAddr	
netErrWouldBlock	

## Net Library

### Net Library Functions

---

**Sockets Equivalent**    `int sendmsg (int socket, const struct msghdr *pbP,  
int flags);`

**Comments**    This call attempts to write data to the given socket and returns the number of bytes actually sent, which may be less than or equal to the requested number of bytes. The data is passed in the scatter-write array specified in the pbP parameter block.

For datagram sockets, you must only send a single packet at a time. If the data is too large to fit in a single UDP packet, no data is sent and -1 is returned.

The toAddrP field applies only to datagram sockets without an existing connection. An error is returned if the datagram socket was previously connected and toAddrP is specified. Stream-based sockets, by definition, must have a connection established with a remote host before data can be written. Raw sockets (supported in Palm OS version 3.0 and higher) must construct the entire IP header, including the destination address, before data can be sent; thus, the address is taken from the data to be sent.

If there isn't enough buffer space to send any data, this call will block until there is space, or until a timeout.

---

**NOTE:** For stream-based sockets, this call may write only a portion of the desired data. It always returns the number of bytes actually written. Consequently, the caller should be prepared to call this routine repeatedly until the desired number of bytes have been written, or until it returns 0 or -1.

---

**See Also**    [NetLibSend](#), [NetLibReceive](#), [NetLibReceivePB](#),  
[NetLibDmReceive](#)

## NetLibSettingGet

<b>Purpose</b>	Retrieves a general setting.										
<b>Declared In</b>	NetMgr.h										
<b>Prototype</b>	<pre>Err NetLibSettingGet (UInt16 libRefNum,                       UInt16 setting, void *valueP, UInt16 *valueLenP)</pre>										
<b>Parameters</b>	<table><tr><td>-&gt; libRefNum</td><td>Reference number of the net library.</td></tr><tr><td>-&gt; setting</td><td>Setting to retrieve, one of the NetSettingEnum constants.</td></tr><tr><td>&lt;- valueP</td><td>Space for return value of setting.</td></tr><tr><td>&lt;-&gt; valueLenP</td><td>On entry, size of valueP. On exit, actual size of setting.</td></tr></table>	-> libRefNum	Reference number of the net library.	-> setting	Setting to retrieve, one of the NetSettingEnum constants.	<- valueP	Space for return value of setting.	<-> valueLenP	On entry, size of valueP. On exit, actual size of setting.		
-> libRefNum	Reference number of the net library.										
-> setting	Setting to retrieve, one of the NetSettingEnum constants.										
<- valueP	Space for return value of setting.										
<-> valueLenP	On entry, size of valueP. On exit, actual size of setting.										
<b>Result</b>	Returns one of the following values: <table><tr><td>0</td><td>Success.</td></tr><tr><td>netErrUnknownSetting</td><td>Invalid setting constant</td></tr><tr><td>netErrPrefNotFound</td><td>No current value for setting</td></tr><tr><td>netErrBufTooSmall</td><td>valueP was too small to hold entire setting. Setting value was truncated to fit in valueP.</td></tr><tr><td>netErrBufWrongSize</td><td></td></tr></table>	0	Success.	netErrUnknownSetting	Invalid setting constant	netErrPrefNotFound	No current value for setting	netErrBufTooSmall	valueP was too small to hold entire setting. Setting value was truncated to fit in valueP.	netErrBufWrongSize	
0	Success.										
netErrUnknownSetting	Invalid setting constant										
netErrPrefNotFound	No current value for setting										
netErrBufTooSmall	valueP was too small to hold entire setting. Setting value was truncated to fit in valueP.										
netErrBufWrongSize											
<b>Sockets Equivalent</b>	None										
<b>Comments</b>	This call retrieves the current value of any general setting. The caller must pass a pointer to a buffer to hold the return value (valueP), the size of the buffer (*valueLenP), and the setting ID (setting). The setting ID is one of the NetSettingEnum constants in the netSettingEnum type.										

## **Net Library**

### *Net Library Functions*

---

Some settings, such as the host table, are variable size. For these types of settings, you can obtain the actual size required for the buffer by passing 0 for `*valueLenP`. The required size is returned in `valueLenP`.

[Table 61.3](#) lists the general settings and the type of each setting.

**Table 61.3 Net Library General Settings**

<b>netSetting...</b>	<b>Type</b>	<b>Description</b>
ResetAll	void	Used for <a href="#">NetLibSettingSet</a> only. This will clear all other settings to their default values.
PrimaryDNS	UInt32	IP address of primary DNS server. This setting must be set to a non-zero IP address in order to support any of the name lookup calls.
SecondaryDNS	UInt32	IP address of secondary DNS server. Set to 0 to have stack ignore this setting.
DefaultRouter	UInt32	IP address of default router. Default value is 0 which is appropriate for most implementations with only one attached interface (besides loopback). Packets with destination IP addresses that don't lie in the subnet of an attached interface will be sent to this router through the default interface specified by the netSettingDefaultIFCreator/ netSettingDefaultIFInstance pair.
DefaultIFCreator	UInt32	Creator of the default network interface. Default value is 0, which is appropriate for most implementations. Packets with destination IP addresses that don't lie in the subnet of a directly attached interface are sent through this interface. If this setting is 0, the stack automatically makes the first non-loopback interface the default interface.
DefaultIFInstance	UInt16	Instance number of the default network interface. Packets with destination IP addresses that don't lie in the subnet of an attached interface are sent through the default interface. Default value is 0.
HostName	Char []	A zero-terminated character string of 64 bytes or less containing the host name of this machine. This setting is not actually used by the stack. It's present mainly for informative purposes and to support the gethostname/sethostname sockets API calls. To clear the host name, call <a href="#">NetLibIFSettingSet</a> with a valueLen of 0.

**Table 61.3 Net Library General Settings (*continued*)**

<b>netSetting...</b>	<b>Type</b>	<b>Description</b>
DomainName	Char []	A zero-terminated character string of 256 bytes or less containing the default domain. This default domain name is appended to all host names before name lookups are performed. If the name is not found, the host name is looked up again without appending the domain name to it. To have the stack not use the domain name, call <a href="#">NetLibIFSettingSet</a> with a valueLen of 0.
HostTbl	Char []	A null-terminated character string containing the host table. This table is consulted first before sending a DNS query to the DNS server(s). To have the stack not use a host table, call <a href="#">NetLibIFSettingSet</a> with a valueLen of 0. The format of a host table is a series of lines separated by '\n' in the following format:  host.company.com A 111.222.333.444
CloseWaitTime	UInt32	The close-wait time in milliseconds. This setting must be specified. See the discussion of the <a href="#">NetLibClose</a> call for an explanation of the close-wait time.
TraceBits	UInt32	A bitfield of various trace bits. See " <a href="#">Tracing Bits</a> ." Default value is (netTracingErrors   netTracingAppMsgs). An application can get a list of events in the trace buffer using the <a href="#">NetLibMaster</a> call.
TraceSize	UInt32	Maximum trace buffer size in bytes. Setting this setting always clears the existing trace buffer. Default is 2 KB.
TraceRoll	UInt8	Boolean value, default is true (non-zero). If true, trace buffer will roll over when it fills. If false, tracing will stop as soon as trace buffer fills.

**See Also** [NetLibSettingSet](#), [NetLibIFSettingSet](#),  
[NetLibIFSettingGet](#), [NetLibMaster](#)

## NetLibSettingSet

**Purpose** Sets a general setting.

**Declared In** NetMgr.h

**Prototype** Err NetLibSettingSet (UInt16 libRefNum,  
                  UInt16 setting, void \*valueP, UInt16 valueLen)

**Parameters**

-> libRefNum	Reference number of the net library.
-> setting	Setting to set; one of the <code>NetSettingEnum</code> constants. See <a href="#">Table 61.3</a> .
-> valueP	New value for the setting.
-> valueLen	Size of new setting.

**Result** Returns one of the following values:

0 Success.

netErrUnknownSetting  
                  Invalid setting constant.

netErrInvalidSettingSize  
                  valueLen was invalid for the given setting.

netErrBufWrongSize

netErrReadOnlySetting

**Sockets Equivalent**  
None

**Comments** This call can be used to set the current value of any general setting. The caller must pass a pointer to a buffer which holds the new value (`valueP`), the size of the buffer (`valueLen`), and the setting ID (`setting`). The setting ID is one of the `NetSettingXXX` constants in the `NetSettingEnum` type.

If the net library is not open at the time this call is made, the setting is stored in the active configuration. You need to save the active configuration using [`NetLibConfigSaveAs`](#) if you want the new

value of the setting to be used the next time the net library is opened.

See [NetLibSettingGet](#) for an explanation of each of the settings.

Of particular interest is the `netSettingResetAll` setting, which, if used, will reset all general settings to their default values. When using this setting, `valueP` and `valueLen` are ignored.

**See Also** [NetLibSettingGet](#), [NetLibSettingSet](#),  
[NetLibIFSettingSet](#), [NetLibMaster](#)

## NetLibSocketAccept

**Purpose** Accept a connection from a stream-based socket.

**Declared In** NetMgr.h

**Prototype** Int16 NetLibSocketAccept (UInt16 libRefnum,  
NetSocketRef socket,  
NetSocketAddrType \*sockAddrP, Int16 \*addrLenP,  
Int32 timeout, Err \*errP)

**Parameters**

-> libRefNum	Reference number of the net library.
-> socket	Descriptor for the open socket.
<- sockAddrP	Address of remote host is returned here.
<->addrLenP	On entry, length of <code>sockAddrP</code> buffer in bytes. On exit, length of returned address stored in <code>*sockAddrP</code> .
-> timeout	Maximum timeout in system ticks; -1 means wait forever.
<- errP	Contains an error code if the return value is -1.

**Result** Returns the `NetSocketRef` of the new socket. If the return value is -1, an error has occurred, and `errP` contains one of the following values:

0 No error.

netErrTimeout Call timed out.  
netErrNotOpen The referenced net library has not been opened yet.  
netErrParamErr  
netErrSocketNotOpen  
netErrSocketNotConnected  
netErrSocketClosedByRemote  
netErrWrongSocketType  
netErrSocketNotListening  
netErrUnimplemented

**Sockets  
Equivalent**

int accept (int socket, void \*sockAddrP,  
int \*addrLenP);

**Comments**

Accepts the next connection request from a remote client. This call is only applicable to stream-based sockets. Before calling NetLibSocketAccept on a socket, a server application needs to:

- Open the socket ([NetLibSocketOpen](#)).
- Bind the socket to a local address ([NetLibSocketBind](#)).
- Set the maximum pending connection-request queue length ([NetLibSocketListen](#)).

NetLibSocketAccept will block until a successful connection request is obtained from a remote client. After a successful connection is made, this call returns with the address of the remote host in \*sockAddrP and the socket descriptor of a new socket as the return value. You then use the new socket to send and receive data.

**See Also**

[NetLibSocketBind](#), [NetLibSocketListen](#)

## NetLibSocketAddr

<b>Purpose</b>	Returns the local and remote addresses currently associated with a socket.																
<b>Declared In</b>	NetMgr.h																
<b>Prototype</b>	<pre>Int16 NetLibSocketAddr (UInt16 libRefnum,                         NetSocketRef socketRef,                         NetSocketAddrType *locAddrP, Int16 *locAddrLenP,                         NetSocketAddrType *remAddrP, Int16 *remAddrLenP,                         Int32 timeout, Err *errP)</pre>																
<b>Parameters</b>	<table><tr><td>-&gt; libRefNum</td><td>Reference number of the net library.</td></tr><tr><td>-&gt; socketRef</td><td>Descriptor for the open socket.</td></tr><tr><td>&lt;- locAddrP</td><td>Local address of socket is returned here.</td></tr><tr><td>&lt;-&gt;locAddrLenP</td><td>On entry, length of locAddrP buffer in bytes. On exit, length of returned address stored in *locAddrP.</td></tr><tr><td>&lt;- remAddrP</td><td>Address of remote host is returned here.</td></tr><tr><td>&lt;-&gt;remAddrLenP</td><td>On entry, length of remAddrP buffer in bytes. On exit, length of returned address stored in *remAddrP.</td></tr><tr><td>-&gt; timeout</td><td>Maximum timeout in system ticks; -1 means wait forever.</td></tr><tr><td>&lt;- errP</td><td>Contains an error code if the return value is -1.</td></tr></table>	-> libRefNum	Reference number of the net library.	-> socketRef	Descriptor for the open socket.	<- locAddrP	Local address of socket is returned here.	<->locAddrLenP	On entry, length of locAddrP buffer in bytes. On exit, length of returned address stored in *locAddrP.	<- remAddrP	Address of remote host is returned here.	<->remAddrLenP	On entry, length of remAddrP buffer in bytes. On exit, length of returned address stored in *remAddrP.	-> timeout	Maximum timeout in system ticks; -1 means wait forever.	<- errP	Contains an error code if the return value is -1.
-> libRefNum	Reference number of the net library.																
-> socketRef	Descriptor for the open socket.																
<- locAddrP	Local address of socket is returned here.																
<->locAddrLenP	On entry, length of locAddrP buffer in bytes. On exit, length of returned address stored in *locAddrP.																
<- remAddrP	Address of remote host is returned here.																
<->remAddrLenP	On entry, length of remAddrP buffer in bytes. On exit, length of returned address stored in *remAddrP.																
-> timeout	Maximum timeout in system ticks; -1 means wait forever.																
<- errP	Contains an error code if the return value is -1.																
<b>Result</b>	Returns 0 upon success and -1 if an error occurred. If the return value is -1, errP contains one of the following values: <table><tr><td>0</td><td>No error.</td></tr><tr><td>netErrTimeout</td><td>Call timed out.</td></tr><tr><td>netErrNotOpen</td><td>The referenced net library has not been opened yet.</td></tr><tr><td>netErrParamErr</td><td></td></tr></table>	0	No error.	netErrTimeout	Call timed out.	netErrNotOpen	The referenced net library has not been opened yet.	netErrParamErr									
0	No error.																
netErrTimeout	Call timed out.																
netErrNotOpen	The referenced net library has not been opened yet.																
netErrParamErr																	

netErrSocketNotOpen  
netErrSocketClosedByRemote  
netErrOutOfCmdBlocks

**Sockets Equivalent**

```
int getpeername (int s, struct sockaddr *name,  
int *namelen);  
  
int getsockname (int s, struct sockaddr *name,  
int *namelen);
```

**Comments** This call is mainly useful for stream-based sockets. It allows the caller to find out what address was bound to a connected socket and the address of the remote host that it's connected to.  
  
In Palm OS version 3.0 and higher, if you pass a raw socket to this function, it returns the instance number and creator of the interface to which the socket is bound.

**See Also** [NetLibSocketBind](#), [NetLibSocketConnect](#),  
[NetLibSocketAccept](#)

## NetLibSocketBind

**Purpose** Assign a local address to a socket.

**Declared In** NetMgr.h

**Prototype**

```
Int16 NetLibSocketBind (UInt16 libRefnum,  
NetSocketRef socket,  
NetSocketAddressType *sockAddrP, Int16 addrLen,  
Int32 timeout, Err *errP)
```

**Parameters**

-> libRefNum	Reference number of the net library.
-> socket	Descriptor for the open socket.
-> sockAddrP	Pointer to the address to give to the socket. This can be a <a href="#">NetSocketAddressINTType</a> or a <a href="#">NetSocketAddressRawType</a> .
-> addrLen	Length of address in *sockAddrP.

## Net Library

### Net Library Functions

---

-> timeout      Maximum timeout in system ticks; -1 means wait forever.

<- errP      Contains an error code if the return value is -1.

**Result**      Returns 0 upon success and -1 if an error occurred. If an error occurred, errP contains one of the following values:

0      No error.

netErrTimeout      Call timed out.

netErrNotOpen      The referenced net library has not been opened yet.

netErrParamErr

netErrSocketNotOpen

netErrSocketAlreadyConnected

netErrSocketClosedByRemote

netErrOutOfCmdBlocks

**Sockets Equivalent**      `int bind (int socket, const void *sockAddrP,  
int addrLen);`

**Comments**      Applications that want to wait for an incoming connection request from a remote host must call this function. After calling NetLibSocketBind, applications can call [NetLibSocketListen](#) and then [NetLibSocketAccept](#) to make the socket ready to accept connection requests.

**Compatibility**      Raw sockets are only supported in Palm OS version 3.0 and higher. See [NetLibSocketOpen](#) for instructions on how to bind raw sockets.

**See Also**      [NetLibSocketConnect](#), [NetLibSocketListen](#),  
[NetLibSocketAccept](#)

## NetLibSocketClose

<b>Purpose</b>	Close a socket.												
<b>Declared In</b>	NetMgr.h												
<b>Prototype</b>	<code>Int16 NetLibSocketClose (UInt16 libRefnum, NetSocketRef socket, Int32 timeout, Err *errP)</code>												
<b>Parameters</b>	<table><tr><td>-&gt; libRefNum</td><td>Reference number of the net library.</td></tr><tr><td>-&gt; socket</td><td>Descriptor for the open socket.</td></tr><tr><td>-&gt; timeout</td><td>Maximum timeout in system ticks; -1 means wait forever.</td></tr><tr><td>&lt;- errP</td><td>Contains an error code if the return value is -1.</td></tr></table>	-> libRefNum	Reference number of the net library.	-> socket	Descriptor for the open socket.	-> timeout	Maximum timeout in system ticks; -1 means wait forever.	<- errP	Contains an error code if the return value is -1.				
-> libRefNum	Reference number of the net library.												
-> socket	Descriptor for the open socket.												
-> timeout	Maximum timeout in system ticks; -1 means wait forever.												
<- errP	Contains an error code if the return value is -1.												
<b>Result</b>	Returns 0 upon success and -1 if an error occurred. If an error occurred, errP contains one of the following values:  <table><tr><td>0</td><td>No error.</td></tr><tr><td>netErrTimeout</td><td>Call timed out.</td></tr><tr><td>netErrNotOpen</td><td>The referenced net library has not been opened yet.</td></tr><tr><td>netErrParamErr</td><td></td></tr><tr><td>netErrSocketNotOpen</td><td></td></tr><tr><td>netErrOutOfCmdBlocks</td><td></td></tr></table>	0	No error.	netErrTimeout	Call timed out.	netErrNotOpen	The referenced net library has not been opened yet.	netErrParamErr		netErrSocketNotOpen		netErrOutOfCmdBlocks	
0	No error.												
netErrTimeout	Call timed out.												
netErrNotOpen	The referenced net library has not been opened yet.												
netErrParamErr													
netErrSocketNotOpen													
netErrOutOfCmdBlocks													
<b>Sockets Equivalent</b>	<code>int close (int socket);</code>												
<b>Comments</b>	Closes down a socket and frees all memory associated with it.												
<b>See Also</b>	<a href="#">NetLibSocketOpen</a> , <a href="#">NetLibSocketShutdown</a>												

## NetLibSocketConnect

<b>Purpose</b>	Assign a destination address to a socket and initiate three-way handshake if it's stream based.																		
<b>Declared In</b>	NetMgr.h																		
<b>Prototype</b>	<pre>Int16 NetLibSocketConnect (UInt16 libRefnum,                            NetSocketRef socket,                            NetSocketAddrType *sockAddrP, Int16 addrLen,                            Int32 timeout, Err *errP)</pre>																		
<b>Parameters</b>	<table><tr><td>-&gt; libRefNum</td><td>Reference number of the net library.</td></tr><tr><td>-&gt; socket</td><td>Descriptor for the open socket.</td></tr><tr><td>-&gt; sockAddrP</td><td>Pointer to address to connect to.</td></tr><tr><td>-&gt; addrLen</td><td>Length of address in *sockAddrP.</td></tr><tr><td>-&gt; timeout</td><td>Maximum timeout in system ticks; -1 means wait forever.</td></tr><tr><td>&lt;- errP</td><td>Contains an error code if the return value is -1.</td></tr></table>	-> libRefNum	Reference number of the net library.	-> socket	Descriptor for the open socket.	-> sockAddrP	Pointer to address to connect to.	-> addrLen	Length of address in *sockAddrP.	-> timeout	Maximum timeout in system ticks; -1 means wait forever.	<- errP	Contains an error code if the return value is -1.						
-> libRefNum	Reference number of the net library.																		
-> socket	Descriptor for the open socket.																		
-> sockAddrP	Pointer to address to connect to.																		
-> addrLen	Length of address in *sockAddrP.																		
-> timeout	Maximum timeout in system ticks; -1 means wait forever.																		
<- errP	Contains an error code if the return value is -1.																		
<b>Result</b>	Returns 0 upon success and -1 if an error occurred. If an error occurred, errP contains one of the following values: <table><tr><td>0</td><td>No error.</td></tr><tr><td>netErrTimeout</td><td>Call timed out.</td></tr><tr><td>netErrNotOpen</td><td>The referenced net library has not been opened yet.</td></tr><tr><td>netErrParamErr</td><td></td></tr><tr><td>netErrSocketNotOpen</td><td></td></tr><tr><td>netErrSocketBusy</td><td></td></tr><tr><td>netErrNoInterfaces</td><td>Incorrect setup.</td></tr><tr><td>netErrPortInUse</td><td></td></tr><tr><td>netErrQuietTimeNotElapsed</td><td></td></tr></table>	0	No error.	netErrTimeout	Call timed out.	netErrNotOpen	The referenced net library has not been opened yet.	netErrParamErr		netErrSocketNotOpen		netErrSocketBusy		netErrNoInterfaces	Incorrect setup.	netErrPortInUse		netErrQuietTimeNotElapsed	
0	No error.																		
netErrTimeout	Call timed out.																		
netErrNotOpen	The referenced net library has not been opened yet.																		
netErrParamErr																			
netErrSocketNotOpen																			
netErrSocketBusy																			
netErrNoInterfaces	Incorrect setup.																		
netErrPortInUse																			
netErrQuietTimeNotElapsed																			

netErrInternal  
netErrSocketAlreadyConnected  
netErrSocketClosedByRemote  
netErrTooManyTCPConnections  
netErrWouldBlock  
netErrWrongSocketType  
netErrOutOfCmdBlocks

**Sockets Equivalent** int connect (int socket, const void \*sockAddrP,  
int addrLen);

**See Also** [NetLibSocketBind](#), [NetUTCPOpen](#)

## NetLibSocketListen

**Purpose** Put a stream-based socket into passive listen mode.

**Declared In** NetMgr.h

**Prototype** Int16 NetLibSocketListen (UInt16 libRefnum,  
NetSocketRef socket, UInt16 queueLen,  
Int32 timeout, Err \*errP)

**Parameters**

-> libRefNum	Reference number of the net library.
-> socket	Descriptor for the open socket.
-> queueLen	Maximum number of pending connections allowed.
-> timeout	Maximum timeout in system ticks; -1 means wait forever.
<- errP	Contains an error code if the return value is -1.

**Result** Returns 0 upon success and -1 if an error occurred. If an error occurred, errP contains one of the following values:

0 No error.

## Net Library

### Net Library Functions

---

netErrTimeout Call timed out.  
netErrNotOpen The referenced net library has not been opened yet.  
netErrParamErr  
netErrOutOfResources  
netErrSocketNotOpen  
netErrSocketBusy  
netErrNoInterfaces  
Incorrect setup.  
netErrPortInUse  
netErrInternal  
netErrSocketAlreadyConnected  
netErrSocketClosedByRemote  
netErrWrongSocketType  
netErrQuietTimeNotElapsed  
netErrOutOfCmdBlocks

#### Sockets Equivalent

int listen (int socket, int queueLen);

#### Comments

Sets the maximum allowable length of the queue for pending connections. This call is only applicable to stream-based (TCP/IP) sockets.

After a socket is created and bound to a local address using [NetLibSocketBind](#), a server application can call [NetLibSocketListen](#) and then [NetLibSocketAccept](#) to accept connections from remote clients.

The queueLen is currently quietly limited to 1 (higher values are ignored).

#### See Also

[NetLibSocketBind](#), [NetLibSocketAccept](#)

## NetLibSocketOpen

**Purpose** Open a new socket.

**Declared In** NetMgr.h

**Prototype** `NetSocketRef NetLibSocketOpen (UInt16 libRefnum,  
NetSocketAddrEnum domain, NetSocketTypeEnum type,  
Int16 protocol, Int32 timeout, Err *errP)`

**Parameters**

-> libRefNum	Reference number of the net library.
-> domain	Address domain. See <a href="#">NetSocketAddrEnum</a> .
-> type	Desired type of connection. See <a href="#">NetSocketTypeEnum</a> .
-> protocol	Protocol to use. This parameter is currently ignored. For raw sockets in the <code>netSocketAddrINET</code> domain, specify one of the following: <code>netSocketProtoIPTCP</code> <code>netSocketProtoIPUDP</code> <code>netSocketProtoIPRAW</code> For all other socket types or for raw sockets in the raw domain, this parameter is ignored.
-> timeout	Maximum timeout in system ticks; -1 means wait forever.
<- errP	Contains an error code if the return value is -1.

**Result** Returns the `NetSocketRef` of the opened socket or -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

0	No error.
<code>netErrTimeout</code>	Call timed out.
<code>netErrNotOpen</code>	The referenced net library has not been opened yet.

## Net Library

### Net Library Functions

---

```
netErrParamErr  
netErrNoMoreSockets  
netErrOutOfCmdBlocks  
netErrOutOfMemory
```

**Sockets Equivalent**    int socket (int domain, int type, int protocol);

#### Comments

Allocates memory for a new socket and opens it.

Raw sockets are supported in Palm OS version 3.0 and higher. Two types of raw sockets are supported:

- Raw sockets in the netSocketAddrINET domain

In this case, you must bind the socket to an IP address using [NetLibSocketBind](#), passing a [NetSocketAddrINType](#) structure for the socket address. The port field is ignored.

For applications that use raw sockets in the INET domain, the net library checks the destination IP address of all incoming packets to see if it matches any of those raw sockets. If it does, the packet is enqueued directly into the matching socket and is **not** passed to the protocol stack.

When an application sends data through raw sockets in the IP domain, the net library packages the data into a packet and passes it directly to the interface's send routine. You are responsible for forming the entire IP header, including any necessary checksums, source and destination IP address, and so on.

- Raw sockets in the netSocketAddrRaw domain with no protocol

In this case, you must bind the socket to an interface using [NetLibSocketBind](#), passing a [NetSocketAddrRawType](#) structure for the socket address. The instance and creator specify which interface the caller wants to receive raw packets from.

When an interface is bound to a raw socket with no protocol, the net library places that interface into raw mode. In raw

mode, the interface passes all incoming packets, no matter what the link layer protocol, to its raw receive function.

When an application sends data through a raw socket with no protocol, the net library packages the data into a packet and passes it directly to the interface's send routine.

The interface remains in raw mode until the raw socket is closed.

**Compatibility** Raw sockets supported only in Palm OS version 3.0 and higher.

**See Also** [NetLibSocketClose](#), [NetUTCPOpen](#)

## NetLibSocketOptionGet

**Purpose** Retrieves the current value of a socket option.

**Declared In** NetMgr.h

**Prototype**

```
Int16 NetLibSocketOptionGet (UInt16 libRefnum,  
    NetSocketRef socket, UInt16 level, UInt16 option,  
    void *optValueP, UInt16 *optValueLenP,  
    Int32 timeout, Err *errP)
```

**Parameters**

-> libRefNum	Reference number of the net library.
-> socket	Descriptor for the open socket.
-> level	Level of the option, one of the NetSocketOptLevelEnum constants. See <a href="#">NetLibSocketOptionSet</a> .
-> option	One of the NetSocketOptEnum constants. See <a href="#">NetLibSocketOptionSet</a> .
<- optValueP	Pointer to variable holding new value of option.
<-> optValueLenP	Size of variable pointed to by optValueP on entry. Actual size of return value on exit.

## Net Library

### Net Library Functions

---

-> timeout      Maximum timeout in system ticks; -1 means wait forever.

<- errP      Contains an error code if the return value is -1.

**Result**      Returns 0 upon success and -1 if an error occurred. If an error occurred, errP contains one of the following values:

0      No error.

netErrTimeout      Call timed out.

netErrNotOpen      The referenced net library has not been opened yet.

netErrParamErr

netErrSocketNotOpen

netErrUnimplemented

netErrWrongSocketType

netErrInvalidSettingSize

**Sockets Equivalent**      `int getsockopt (int socket, int level, int option,  
const void *optValueP, int *optValueLenP);`

**Comments**      Returns the current value of a socket option. The caller passes a pointer to a variable to hold the returned value (in optValueP) and the size of this variable (in \*optValueLenP). On exit, \*optValueP is updated with the actual size of the return value.

For all of the fixed size options (every option except netSocketOptIPOptions), \*optValueLenP is unmodified on exit and this call does its best to return the value in the caller's desired type size.

For compatibility with existing Internet applications, this call is quite flexible on the \*optValueLenP parameter. If the desired type for an option is FLAG, this call supports an \*optValueLenP of 1, 2, or 4. If the desired type for an option is int, it supports an \*optValueLenP of 2 or 4.

See [NetLibSocketOptionSet](#) for a list of available options.

**See Also**      [NetLibSocketOptionSet](#)

## NetLibSocketOptionSet

**Purpose** Set a socket option.

**Declared In** NetMgr.h

**Prototype** `Int16 NetLibSocketOptionSet (UInt16 libRefnum,  
NetSocketRef socket, UInt16 level, UInt16 option,  
void *optValueP, UInt16 optValueLen,  
Int32 timeout, Err *errP)`

**Parameters**

-> libRefNum	Reference number of the net library.
-> socket	Descriptor for the open socket.
-> level	Level of the option, one of the NetSocketOptLevelEnum constants. See the comments section.
-> option	One of the NetSocketOptEnum constants. See the comments section.
-> optValueP	Pointer to the variable holding the new value of the option.
-> optValueLen	Size of variable pointed to by optValueP.
-> timeout	Maximum timeout in system ticks; -1 means wait forever.
<- errP	Contains an error code if the return value is -1.

**Result** Returns 0 upon success and -1 if an error occurred. If an error  
occurred, errP contains one of the following values:

0	No error.
netErrTimeout	Call timed out.
netErrNotOpen	The referenced net library has not been opened yet.
netErrParamErr	
netErrSocketNotOpen	
netErrUnimplemented	

netErrWrongSocketType  
netErrInvalidSettingSize

**Sockets Equivalent**    int setsockopt (int socketRef, int level,  
                          int option, const void \*optValueP,  
                          int optValueLen);

**Comments**    Sets various options associated with a socket. The caller passes a pointer to the new option value in optValueP and the size of the option in optValueLen.

[Table 61.4](#) lists the available options.

- The Level column specifies the option level, which is one of the netSocketOptLevelXXX constants.
- The Option column lists the option, which is one of the netSocketOptXXX constants.
- The G/S column lists whether this option can be fetched with the [NetLibSocketOptionGet](#) call (G) and/or set (S) with this call.
- The type column lists the data type of the option.
- The I column specifies whether or not this option is currently implemented.

**Table 61.4 Net Library Socket Options**

netSocketOptLevel...	netSocketOpt...	G/S	Type	I	Description
IP	IPOptions	GS	UInt8 []	N	Options in IP Header
TCP	TCPNoDelay	GS	FLAG	Y	Don't delay send to coalesce packets
TCP	TCPMaxSeg	G	int	Y	Get TCP maximum segment size
Socket	SockDebug	GS	FLAG	N	Turn on recording of debug info
Socket	SockAcceptConn	G	FLAG	N	Socket has had listen

**Table 61.4 Net Library Socket Options (*continued*)**

<b>netSocket OptLevel...</b>	<b>netSocketOpt...</b>	<b>G/S</b>	<b>Type</b>	<b>I</b>	<b>Description</b>
Socket	SockReuseAddr	GS	FLAG	N	Allow local address reuse
Socket	SockKeepAlive	GS	FLAG	Y	Keep connections alive
Socket	SockDontRoute	GS	FLAG	N	Just use interface addresses
Socket	SockBroadcast	GS	FLAG	N	Permit sending of broadcast messages
Socket	SockUseLoopback	GS	FLAG	N	Bypass hardware when possible
Socket	SockLinger	GS	NetSocketLingerType	Y	Linger on close if data present NetSocketLingerType is a structure with two fields: onOff (true or false) and time (linger time in seconds).
Socket	SockOOBInLine	GS	FLAG	N	Leave received OOB data in-line
Socket	SockSndBufSize	GS	int	N	Send buffer size
Socket	SockRcvBufSize	GS	int	N	Receive buffer size
Socket	SockSndLowWater	GS	int	N	Send low-water mark
Socket	SockRcvLowWater	GS	int	N	Receive low-water mark
Socket	SockSndTimeout	GS	int	N	Send timeout
Socket	SockRcvTimeout	GS	int	N	Receive timeout
Socket	SockErrorStatus	G	int	Y	Get error status and clear
Socket	SockSocketType	G	int	Y	Get socket type

## Net Library

### *Net Library Functions*

---

**Table 61.4 Net Library Socket Options (*continued*)**

<b>netSocket OptLevel...</b>	<b>netSocketOpt...</b>	<b>G/S</b>	<b>Type</b>	<b>I</b>	<b>Description</b>
Socket	SockNonBlocking	GS	FLAG	Y	Set non-blocking mode on/off
Socket	SockRequireErrClear	GS	FLAG	Y	Return the current error status for all subsequent socket function calls until the error is cleared.

For compatibility with existing Internet applications, this call is quite flexible on the optValueLen parameter. If the desired type for an option is FLAG, this call accepts an optValueLen of 1, 2, or 4. If the desired type for an option is int, it accepts an optValueLen of 2 or 4.

Except for the `netSocketOptSockNonBlocking` option, all options listed above have equivalents in the sockets API. The `netSocketOptSockNonBlocking` option was added to this call in the net library in order to implement the functionality of the UNIX `fcntl()` control call, which can be used to turn nonblocking mode on and off for sockets.

**See Also** [NetLibSocketOptionGet](#)

## NetLibSocketShutdown

**Purpose** Shut down a socket in one or both directions.

**Declared In** NetMgr.h

**Prototype** Int16 NetLibSocketShutdown (UInt16 libRefnum,  
NetSocketRef socket, Int16 direction,  
Int32 timeout, Err \*errP)

**Parameters**

-> libRefNum	Reference number of the net library.
-> socket	Descriptor for the open socket.
-> direction	Direction to shut down. One of the <code>NetSocketDirEnum</code> constants. Specifically: <code>netSocketDirInput</code> <code>netSocketDirOutput</code> <code>netSocketDirBoth</code>
-> timeout	Maximum timeout in system ticks; -1 means wait forever.
<- errP	Contains an error code if the return value is -1.

**Result** Returns 0 upon success and -1 if an error occurred. If an error occurred, `errP` contains one of the following values:

0	No error.
<code>netErrTimeout</code>	Call timed out.
<code>netErrNotOpen</code>	The referenced net library has not been opened yet.

## Net Library

### Net Library Functions

---

netErrParamErr  
netErrSocketNotOpen  
netErrNoMultiPktAddr  
netErrOutOfCmdBlocks

**Sockets Equivalent** int shutdown (int socket, int direction);

**Comments** Shuts down communication in one or both directions on a socket. If direction is netSocketDirInput, the socket is marked as down in the receive direction and further read operations from it return a netErrSocketInputShutdown error.

## NetLibTracePrintF

**Purpose** Store debugging information in the net library's trace buffer.

**Declared In** NetMgr.h

**Prototype** Err NetLibTracePrintF (UInt16 libRefNum,  
const Char \*formatStr, ...)

**Parameters** -> libRefNum Reference number of the net library.  
-> formatStr A printf style format string.  
-> ... Arguments to the format string.

**Result** Returns 0 upon success or netErrNotOpen if the net library has not been opened.

**Sockets Equivalent** None

**Comments** This call is a convenient debugging tool for developing Internet applications. It stores a message into the net library's trace buffer, which can later be dumped using the [NetLibMaster](#) call. The net library's trace buffer is used to store run-time errors that the net

library encounters as well as errors and messages from network interfaces and from applications that use this call.

The `formatStr` parameter is a `printf` style format string which supports the following format specifiers:

`%d, %i, %u, %x, %s, %c`

but it does **not** support field widths, leading 0's etc.

Note that the `netTracingAppMsgs` bit of the `netSettingTraceBits` setting must be set using the call `NetLibSettingSet(...netSettingTraceBits...)`. Otherwise, this routine will do nothing.

**See Also** [NetLibTracePutS](#), [NetLibMaster](#), [NetLibSettingSet](#)

## NetLibTracePutS

**Purpose** Store debugging information in the net library's trace buffer.

**Declared In** NetMgr.h

**Prototype** Err NetLibTracePutS (UInt16 libRefNum,  
Char \*strP)

**Parameters** -> libRefNum Reference number of the net library.  
-> strP String to store in the trace buffer.

**Result** Returns 0 upon success or `netErrNotOpen` if the net library has not been opened.

**Sockets Equivalent** None

**Comments** This call is a convenient debugging tool for developing Internet applications. It will store a message into the net library's trace buffer which can later be dumped using the [NetLibMaster](#) call. The net library's trace buffer is used to store run-time errors that the net library encounters as well as errors and messages from network interfaces and from applications that use this call.

## Net Library

### Net Library Functions

---

Note the netTracingAppMsgs bit of the netSettingTraceBits setting must be set using the NetLibSettingSet(...netSettingTraceBits...) call or this routine will do nothing.

**See Also** [NetLibTracePrintF](#), [NetLibMaster](#), [NetLibSettingSet](#)

## NetNToHL

**Purpose** Macro that converts a 32-bit value from network to host byte order.

**Declared In** NetBitUtils.h

**Prototype** NetNToHL (x)

**Parameters** -> x                    32-bit value to convert.

**Result** Returns x in host byte order.

**Errors** none

**Sockets Equivalent** ntohs()

**See Also** [NetNToHS](#), [NetHToNL](#), [NetHToNS](#)

## NetNToHS

**Purpose** Macro that converts a 16-bit value from network to host byte order.

**Declared In** NetBitUtils.h

**Prototype** NetNToHS (x)

**Parameters** -> x                    16-bit value to convert.

**Result** Returns x in host byte order.

**Errors** None

**Sockets  
Equivalent** ntohs()

**See Also** [NetHToNL](#), [NetNToHL](#), [NetHToNS](#)

## **Net Library**

### *Net Library Functions*

---

# Network Utilities

---

This chapter describes network utilities provided in the module `NetSocket.c`. These utilities are convenience functions that you can use in place of net library functions in applications that use the net library. You can find `NetSocket.c` in the folder `Libraries\Net\Src`. (On Palm OS® 3.5, `NetSocket.c` is in the folder `CodeWarrior Libraries\Comms\NetSocket\Src`.)

The include file for the functions described in this chapter is `<unix/sys_socket.h>`. This header file is not included by any other Palm™ header file; you must explicitly include it in your code.

For more information on `NetSocket.c` and `sys_socket.h`, see the chapter “[Network Communication](#)” in the *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## Network Utility Functions

### NetUReadN

**Purpose** Reads a specified number of bytes from a socket.

**Declared In** `unix/sys_socket.h`

**Prototype** `Int32 NetUReadN (NetSocketRef fd, UInt8* bufP, UInt32 numBytes)`

**Parameters**

-> <code>fd</code>	Descriptor for the open socket.
<- <code>bufP</code>	Pointer to buffer to hold received data.
-> <code>numBytes</code>	Number of bytes to read.

**Result** Returns the number of bytes actually read. If the return value is less than 0, an error occurred.

## Network Utilities

### Network Utility Functions

---

**Comments** This function repeatedly calls [NetLibReceive](#) until numBytes have been read or until NetLibReceive returns an error.

**See Also** [NetUWriteN](#)

## NetUTCPOpen

**Purpose** Opens a TCP (streams-based) socket and connects it to a server.

**Declared In** unix/sys\_socket.h

**Prototype** NetSocketRef NetUTCPOpen (Char\* hostName,  
Char\* serviceName, Int16 port)

**Parameters** -> hostName Remote host, given either by name or by dotted decimal address.

-> serviceName The name of a network service. Possible services are "echo", "discard", "daytime", "qotd", "chargen", "ftp-data", "ftp", "telnet", "smtp", "time", "name", "finger", "pop2", "pop3", "nntp", "imap2". The value of this parameter is ignored if the port parameter is greater than zero.

-> port The number of the port to connect to on the remote host. Set port to zero to use serviceName instead.

**Result** Returns the socket descriptor of the socket that was connected, or -1 if an error occurred.

**Comments** If serviceName is given and port is less than or equal to zero, this function looks up the port number for that service on the remote host and uses it for the connection.

This function is the equivalent of calling [NetLibSocketOpen](#) and [NetLibSocketConnect](#) (or socket and connect).

**NOTE:** This function does not return specific reasons for failure if there is a failure. This function is not production-quality code. It is provided as a quick and dirty way of creating a connection and as sample code that can be used as a reference.

---

## NetUWriteN

**Purpose** Writes the specified number of bytes to a socket.

**Declared In** unix/sys\_socket.h

**Prototype** Int32 NetUWriteN (NetSocketRef fd, UInt8\* bufP,  
                  UInt32 numBytes)

**Parameters**

-> fd	Descriptor for the open socket.
-> bufP	Pointer to buffer to write.
-> numBytes	Number of bytes to write.

**Result** Returns the number of bytes actually sent. If the return value is less than 0, an error occurred.

**Comments** This function repeatedly calls [NetLibSend](#) until numBytes have been written or until NetLibSend returns an error.

**See Also** [NetUReadN](#)

## **Network Utilities**

### *Network Utility Functions*

---

# Script Plugin

---

This chapter describes the login script plugin support. You write a plugin to add to the list of available login script commands in the Network preferences panel. This chapter covers:

- [Script Plugin Data Types](#)
- [Script Plugin Constants](#)

The header file `ScriptPlugin.h` declares the API described in this chapter.

For more information on the script plugin, see the section “[Extending the Network Login Script Support](#)” on page 185 in the “[Network Communication](#)” chapter of the *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## Script Plugin Data Types

### **PluginCallbackProcType**

The `PluginCallbackProcType` defines the `procP` field in [`PluginExecCmdType`](#).

```
typedef struct {
    ScriptPluginSelectorProcPtr selectorProcP;
} PluginCallbackProcType,
*PluginCallbackProcPtr;
```

#### **Field Descriptions**

<code>selectorProcP</code>	The address of a selector-based callback function for accessing the functionality of the network interface. Each network interface provides its own <code>ScriptPluginSelectorProc</code> function. See <a href="#">ScriptPluginSelectorProc</a> .
----------------------------	--

## **Script Plugin**

### *Script Plugin Data Types*

---

## **PluginCmdPtr**

The `PluginCmdPtr` type defines a pointer to a [PluginCmdType](#) structure.

```
typedef PluginCmdType * PluginCmdPtr;
```

## **PluginCmdType**

The `PluginCmdType` structure specifies the name of a command.

```
typedef struct {
    Char      commandName [pluginMaxCmdNameLen + 1];
    Boolean   hasTxtStringArg;
    UInt8     reserved;
} PluginCmdType;
```

### **Field Descriptions**

<code>commandName</code>	The name of the command. This string appears in the pull-down list in the Network preferences panel's script view.  The pull-down list contains all available commands from all plugins. Make sure that your command name is unique and as short as possible.
<code>hasTxtStringArg</code>	<code>true</code> if the command takes an argument. In this case when the user selects this command, the Network preferences panel displays a field next to the command name where the user should enter the argument. This argument is passed in the <code>txtStringArg</code> field in <a href="#">PluginExecCmdType</a> when the command is to be executed.
<code>reserved</code>	Reserved for future use.

## PluginExecCmdType

The PluginExecCmdType structure defines the parameter block for the scptLaunchCmdExecuteCmd launch code. This structure specifies which command is to be executed and provides any necessary arguments for the command. Your plugin should respond by executing the command.

```
typedef struct {
    Char      commandName [pluginMaxCmdNameLen + 1];
    Char      txtStringArg
                [pluginMaxLenTxtStringArg +
    1];
    PluginCallbackProcPtr procP;
    void * handle;
} PluginExecCmdType, *PluginExecCmdPtr;
```

### Field Descriptions

commandName	The command's name. This is the string that appears in the pull-down list in the script view of the Network preferences panel.
txtStringArg	If the command takes an argument, this field provides the argument as a string. A NULL value means either that the user did not provide a value, or that you didn't specify that the command takes an argument.
procP	A pointer to a <a href="#">PluginCallbackProcType</a> structure, which identifies the network interface function that the plugin can use to execute the command.
handle	Handle to information specific to a particular connection. You must pass this value when you call the function pointed to by procP.

## **Script Plugin**

### *Script Plugin Data Types*

---

## **PluginInfoPtr**

The `PluginInfoPtr` type defines a pointer to a [PluginInfoType](#) structure.

```
typedef PluginInfoType * PluginInfoPtr;
```

## **PluginInfoType**

The `PluginInfoType` structure is the parameter block for the `scptLaunchCmdListCmds` launch code. When your plugin receives the launch code, the `PluginInfoType` structure is empty. Your plugin should fill in the `PluginInfoType` and return it. The system uses the information returned to construct the pull-down list of available script commands and build a table of which plugin will execute which script command.

```
typedef struct {
    Char pluginName [pluginMaxModuleNameLen + 1];
    UInt16 numOfCommands;
    PluginCmdType command [pluginMaxNumOfCmds];
} PluginInfoType;
```

### **Field Descriptions**

<code>pluginName</code>	A name that the system can use to identify your plugin. This is typically the same name you give the PRC file.
<code>numOfCommands</code>	The number of commands that your plugin defines. The maximum allowed is <code>pluginMaxNumOfCmds</code> .
<code>command</code>	An array of <a href="#">PluginCmdType</a> structures that provide information about the commands that your plugin defines.

## **ScriptPluginLaunchCodesEnum**

The `ScriptPluginLaunchCodesEnum` defines the launch codes for the script plugin. Your script plugin's [PilotMain](#) function should respond to the launch codes defined in this enum.

```
typedef enum {
    scptLaunchCmdDoNothing =
        sysAppLaunchCmdCustomBase,
    scptLaunchCmdListCmds,
    scptLaunchCmdExecuteCmd
} ScriptPluginLaunchCodesEnum;
```

### **Value Descriptions**

<code>scptLaunchCmdDoNothing</code>	This launch code is a no-op supplied only to provide a beginning value for the script plugin launch codes. It is not necessary to respond to this launch code.
<code>scptLaunchCmdListCmds</code>	Provide information about the commands that your plugin executes. See <a href="#">PluginInfoType</a> .
<code>scptLaunchCmdExecuteCmd</code>	Execute the specified command.  This launch code is received when the system is executing a user's login script during a network connection attempt. Your plugin should respond by executing the command provided in the <a href="#">PluginExecCmdType</a> parameter block.

# Script Plugin Constants

## Command Constants

The following constants identify the available commands that the network interface can perform for you. These commands are building blocks that you use to create your own script commands. To perform one of these tasks, pass the constant value as an argument to the network interface's callback function ([ScriptPluginSelectorProc](#)).

Constant	Value	Description
pluginNetLibDoNothing	0	For debugging purposes.
pluginNetLibReadBytes	1	Read the specified number of bytes from the open connection.
pluginNetLibWriteBytes	2	Write the specified number of bytes to the open connection.
pluginNetLibGetUserName	3	Get the user name from the network service profile.
pluginNetLib GetUserPwd	4	Get the user's password from the network service profile.
pluginNetLibCheckCancelStatus	5	Check to see if the user canceled the connection.
pluginNetLibPromptUser	6	Prompt the user for input.
pluginNetLibConnLog	7	Write a string to the network service's connection log.
pluginNetLibCallUIProc	8	Have the network interface call a function in your plugin that displays UI.

Constant	Value	Description
		Use this command if you need to display a more complicated user interface than the simple user prompt that the network interface provides.
pluginNetLibGetSerLibRefNum	9	Obtain the serial library's reference number. You need the reference number to perform any serial library commands, which you might need to perform more complex work with the connection port.

## Size Constants

The following table lists constants that control the size of strings in your plugin and the size of the plugin itself.

Constant	Value	Description
pluginMaxCmdNameLen	15	The maximum length for the command's name, not including the terminating null character. This is the string displayed to the user in the pull-down menu.
pluginMaxModuleNameLen	15	The maximum length for the plugin's name (not including the terminating null character), which is typically the name of the PRC file as well.
pluginMaxNumOfcmds	10	The maximum number of commands that your plugin can define.
pluginMaxLenTxtStringArg	63	The maximum length of the argument that each command can take, not including the terminating null character.

# Script Plugin Functions

## ScriptPluginSelectorProc

**Purpose** A function provided by the network interface for the purpose of performing script commands.

**Declared In** ScriptPlugin.h

**Prototype** Err (\*ScriptPluginSelectorProcPtr) (void \*handle,  
UInt16 command, void \*dataBufferP, UInt16 \*sizeP,  
UInt16 \*dataTimeoutP, void \*procAddrP);

<b>Parameters</b>	-> handle	Handle to information specific to a particular connection.
	-> command	The command to be executed. See “ <a href="#">Command Constants</a> ” for a list of possible values. The rest of the parameters to this callback function are interpreted differently based on the value of the command parameter. See the table in the “Comments” section for specifics.
	<-> dataBufferP	A pointer to arguments to pass to the command or a pointer to data returned by the command. See the “Comments” section below.
	<-> sizeP	The size of dataBufferP.
	-> dataTimeoutP	Number of seconds to wait for the command to execute. 0 means wait forever. Applies only to commands that request information from the network.

-> procAddrP      Pointer to a user interface callback function that the network interface should call to complete the function. Used only by pluginNetLibCallUIProc. This function should take one argument of the same type that you pass to dataBufferP and should return void.

**Result**      Returns 0 upon success, or an error condition upon failure. If an error condition is returned, your plugin should stop processing and return the error condition from its [PilotMain](#).

**Comments**      When your plugin receives the scptLaunchCmdExecuteCmd launch code, the parameter block contains the command's name, its text string argument (if any), and a pointer to the network interface's callback function. You should use this callback function any time you need to communicate with the network library, the user, or the host computer during execution of your command.

The callback function takes as arguments the handle to information about this connection (which is also passed in the launch code's parameter block), and the command that the service should execute. The rest of the parameters are interpreted differently based on what the value the command argument is. See the table below.

pluginNetLib	dataBufferP	sizeP	dataTimeOutP	procAddrP
DoNothing	N/A	N/A	N/A	N/A
ReadBytes	On return, contains the bytes that were read.	On input, contains the number of bytes to read.	Number of seconds to wait before timing out the operation.	N/A
		On return, contains the number of bytes actually read.		

## **Script Plugin**

### *Script Plugin Functions*

<b>pluginNetLib</b>	<b>dataBufferP</b>	<b>sizeP</b>	<b>dataTimeOutP</b>	<b>procAddrP</b>
WriteBytes	On input, contains the data to send.	On input, contains the number of bytes to send.  On return, contains the number of bytes actually sent.	Number of seconds to wait for a response before canceling.	N/A
UserName	On return, contains the user's name	On return, contains the size of the string pointed to by dataBufferP.	N/A	N/A
UserPwd	On return, contains the user's password.	On return, contains the size of the string pointed to by dataBufferP.	N/A	N/A
CheckCancelStatus	On return, the Boolean value true if the user canceled the command, false otherwise.	Size of Boolean.	N/A	N/A
PromptUser	On input, the prompt to display.  On return, the text that the user entered.	On input and on return, the size of the string pointed to by dataBufferP.	N/A	N/A

<b>pluginNetLib</b>	<b>dataBufferP</b>	<b>sizeP</b>	<b>dataTimeOutP</b>	<b>procAddrP</b>
ConnLog	The string that should be written to the log.	N/A	N/A	N/A
CallUIProc	A pointer to a structure to pass to your callback function as a parameter. This structure should contain a handle to the form to be displayed, plus any other necessary information.	N/A	N/A	A pointer to a function in your plugin that displays the form.
GetSerLib RefNum	On return, contains the serial library's reference number.	N/A	N/A	N/A

---

## **Script Plugin**

### *Script Plugin Functions*

---

# Virtual Drivers

---

This chapter provides reference material for the Serial Manager virtual device driver API:

- [Driver Data Structures](#)
- [Driver Constants](#)
- [Virtual Driver-Defined Functions](#)
- [Serial Manager Queue Functions](#)

The header files `SerialVdrv.h` and `SerialDrvrv.h` declare the virtual driver API. For more information on writing device drivers for the Serial Manager, see section “[Writing a Virtual Device Driver](#)” on page 114 in the “[Serial Communication](#)” chapter of *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## Driver Data Structures

### **DrvrInfoType**

The `DrvrInfoType` structure defines information about the serial hardware. It is passed to and filled in by the [`DrvEntryPointProcPtr`](#) for a virtual driver.

```
typedef struct {
    UInt32 drvrID;
    UInt32 drvrVersion;
    UInt32 maxBaudRate;
    UInt32 handshakeThreshold;
    UInt32 portFlags;
    const Char *portDesc;
    DrvrIRQEnum irqType;
    UInt8 multipleEnumerations;
    UInt32 dbCreator;
} DrvrInfoType;
```

## Virtual Drivers

### Driver Data Structures

---

#### Value Descriptions

drvrvID	4-character creator type, such as 'u328'.
drvrvVersion	Version of code that works for this hardware. See <a href="#">Driver Version Constants</a> .
maxBaudRate	Maximum baud rate supported by this hardware.
handshakeThreshold	Baud rate at which the use of hardware handshaking is necessary.
portFlags	Bit flags denoting features of this hardware. The flags are described in <a href="#">Port Feature Constants</a> .
portDesc	Pointer to null-terminated string describing this hardware. This string appears in the Connection panel to describe the port to the user (only if the portCncMgrVisible bit in portFlags is set). Can be NULL if the driver contains a resource (of type 'tSTR' and id kPortDescStrID) that supplies this string.
irqType	IRQ line being used for this hardware. For a virtual driver, specify drvrIRQNone.
multipleEnumerations	The number of entries in the driver table required for this driver. If 0, the driver has a single entry.
dbCreator	Creator ID of the database containing this driver.

#### Compatibility

The multipleEnumerations and dbCreator fields are only defined if [New Serial Manager Feature Set Version 2](#) is present.

## DrvrRcvQType

The DrvrRcvQType structure defines the virtual driver receive buffer and function pointers to functions that access and save data to the buffer. A pointer to this structure is passed to the [VdrvOpenProcPtr](#) function. The DrvrHWRCvQPtr type defines a pointer to a DrvrRcvQType structure.

```
typedef struct DrvrRcvQType {  
    void *rcvQ;  
    WriteByteProcPtr qWriteByte;  
    WriteBlockProcPtr qWriteBlock;  
    GetSizeProcPtr qGetSize;  
    GetSpaceProcPtr qGetSpace;  
    SignalCheckPtr qSignalCheck;  
} DrvrRcvQType;  
  
typedef DrvrRcvQType *DrvrHWRCvQPtr;
```

### Value Descriptions

rcvQ	Pointer to the receive buffer.
qWriteByte	Function pointer to a function that the virtual driver can use to write one byte to the Serial Manager's receive queue. See the <a href="#">WriteByteProcPtr</a> function.
qWriteBlock	Function pointer to a function that the virtual driver can use to write a block of bytes to the Serial Manager's receive queue. See the <a href="#">WriteBlockProcPtr</a> function.
qGetSize	Function pointer to a function that the virtual driver can use to get the total size of the Serial Manager's receive queue. See the <a href="#">GetSizeProcPtr</a> function.

qGetSpace	Function pointer to a function that the virtual driver can use to get the available space in the Serial Manager's receive queue. See the <a href="#">GetSpaceProcPtr</a> function.
qSignalCheck	Function pointer to a function that the virtual driver can use to perform a signal check for the Serial Manager's receive queue. See the <a href="#">SignalCheckPtr</a> function.

**Compatibility** The qSignalCheck field is only defined if [New Serial Manager Feature Set Version 2](#) is present.

## DrvrvStatusEnum

The DrvrvStatusEnum enumerated type specifies serial status bit flags. Return these enumerated types from the [VdrvStatusProcPtr](#) call.

```
typedef enum DrvrvStatusEnum {  
    drvrStatusCtsOn = 0x0001,  
    drvrStatusRtsOn = 0x0002,  
    drvrStatusDsrOn = 0x0004,  
    drvrStatusTxFifoFull = 0x0008,  
    drvrStatusTxFifoEmpty = 0x0010,  
    drvrStatusBreakAsserted = 0x0020,  
    drvrStatusDataReady = 0x0040,  
    drvrStatusLineErr = 0x0080  
} DrvrvStatusEnum;
```

### Value Descriptions

drvrStatusCtsOn	Set if CTS line is active.
drvrStatusRtsOn	Set if RTS line is active.
drvrStatusDsrOn	Set if DSR is on.
drvrStatusTxFifoFull	Set if transmit FIFO is full; cleared if FIFO has space.
drvrStatusTxFifoEmpty	Set if transmit FIFO is empty.

drvrStatusBreakAsserted	Set if sending break characters is enabled.
drvrStatusDataReady	Used by debugger only.
drvrStatusLineErr	Used by debugger only.

## SrmRcvQType

The SrmRcvQType structure defines the Serial Manager receive queue. This queue is passed as a parameter to the virtual driver.

```
typedef struct SrmRcvQType {
    UInt32 qStart;
    UInt32 qEnd;
    UInt32 qSize;
    UInt8 *qData;
    void *qPort;
} SrmRcvQType;
```

### Field Descriptions

qStart	The start of the queue.
qEnd	The end of the queue.
qSize	The size of the queue.
qData	The data currently in the queue.
qPort	A pointer to the current foreground port.

### Compatibility

The SrmRcvQType structure was previously a private structure. It is declared publicly if [New Serial Manager Feature Set Version 2](#) is present.

## VdrvAPIType

The VdrvAPIType structure defines function pointers to the required virtual driver functions. When passed a pointer to this structure in the [DrvEntryPointProcPtr](#) function, that function must fill in the pointers to the virtual driver functions appropriately.

## Virtual Drivers

### Driver Data Structures

---

```
typedef struct {
    VdrvOpenProcPtr drvOpen;
    VdrvCloseProcPtr drvClose;
    VdrvControlProcPtr drvControl;
    VdrvStatusProcPtr drvStatus;
    VdrvReadProcPtr drvRead;
    VdrvWriteProcPtr drvWrite;
    VdrvOpenProcV4Ptr drvOpenV4;
    VdrvControlCustomProcPtr drvControlCustom;
} VdrvAPIType;
```

#### Value Descriptions

drvOpen	Pointer to the driver open function.
drvClose	Pointer to the driver close function.
drvControl	Pointer to the driver control function.
drvStatus	Pointer to the driver status function.
drvRead	Pointer to the driver read function.
drvWrite	Pointer to the driver write function.
drvOpenV4	Pointer to the driver open function for <a href="#">New Serial Manager Feature Set Version 2</a> .
drvControlCustom	Pointer to the driver custom control function.

#### Compatibility

drvOpenV4 and drvControlCustom are declared if both [New Serial Manager Feature Set Version 2](#) and [4.0 New Feature Set](#) are present.

## VdrvConfigType

The VdrvConfigType structure specifies parameters for opening a serial port. This structure is passed as a parameter to [VdrvOpenProcV4Ptr](#).

```
typedef struct VdrvConfigType {
    UInt32 baud;
    UInt32 drvrId;
```

```
    UInt32 function;
    MemPtr drvrDataP;
    UInt16 drvrDataSize;
    UInt32 sysReserved1;
    UInt32 sysReserved2;
} VdrvConfigType;
```

### Field Descriptions

baud	Baud rate at which to open the connection. Serial drivers that do not require baud rates ignore this field.
drvrId	Creator ID of the application or library that is using the Serial Manager.
function	The reason why the port was opened. Specify the creator ID of the application that is opening the port or one of the following values:
serFncUndefined	Undefined function. This is the default value for this field.
serFncPPPSession	The connection is to be used for the PPP interface.
serFncSLIPSession	The connection is to be used for the SLIP session.
serFncDebugger	The connection is to be used for a debugging session.
serFncHotSync	The connection is to be used for a HotSync operation.
serFncConsole	The connection is to the debugging console.
serFncTelephony	The connection is to the telephony library.

The function field is used by protocols such as USB and Bluetooth that perform different setup tasks based on which type of application is using them. RS-232 drivers ignore this parameter.

drvrdatap Pointer to a driver-specific data block.

drvrdatasize The size of the data block pointed to by drvrdatap.

sysreserved1 Reserved for future use.

sysreserved2 Reserved for future use.

**Compatibility** This structure is only defined if both [New Serial Manager Feature Set Version 2](#) and [4.0 New Feature Set](#) are present.

## VdrvCtlOpCodeEnum

The VdrvCtlOpCodeEnum enumerated type specifies a serial control operation. You should handle each of these constants when passed for the controlCode parameter to the [VdrvControlProcPtr](#) call.

```
typedef enum VdrvCtlOpCodeEnum {
    vdrvOpCodeNoOp = 0,
    vdrvOpCodeSetBaudRate = 0x1000,
    vdrvOpCodeSetSettingsFlags,
    vdrvOpCodeSetCtsTimeout,
    vdrvOpCodeClearErr,
    vdrvOpCodeSetSleepMode,
    vdrvOpCodeSetWakeupMode,
    vdrvOpCodeFIFOCount,
    vdrvOpCodeStartBreak,
    vdrvOpCodeStopBreak,
    vdrvOpCodeStartLoopback,
    vdrvOpCodeStopLoopback,
    vdrvOpCodeFlushTxFIFO,
    vdrvOpCodeFlushRxFIFO,
    vdrvOpCodeSendBufferData,
    vdrvOpCodeRcvCheckIdle,
    vdrvOpCodeEmuSetBlockingHook,
```

```
vdrvOpCodeGetOptTransmitSize,  
vdrvOpCodeGetMaxRcvBlockSize,  
vdrvOpCodeNotifyBytesReadFromQ,  
vdrvOpCodeSetDTRAsserted,  
vdrvOpCodeGetDTRAsserted,  
vdrvOpCodeWaitForConfiguration,  
vdrvOpCodeGetUSBDeviceDescriptor,  
vdrvOpCodeGetUSBConfigDescriptor,  
vdrvOpCodeEnableIRDA,  
vdrvOpCodeDisableIRDA,  
vdrvOpCodeEnableUART,  
vdrvOpCodeDisableUART,  
vdrvOpCodeRxEnable,  
vdrvOpCodeRxDisable,  
vdrvOpCodeLineEnable,  
vdrvOpCodeEnableUARTInterrupts,  
vdrvOpCodeDisableUARTInterrupts,  
vdrvOpCodeSetReceiveQueue,  
vdrvOpCodeSaveState,  
vdrvOpCodeRestoreState,  
vdrvOpCodeSetYieldPortCallback,  
vdrvOpCodeSetYieldPortRefCon,  
vdrvOpCodeUserDef = 0x2000,  
vdrvOpCodeSystem = 0x7000,  
vdrvOpCodeCustom = 0x8000  
} VdrvCtlOpCodeEnum;
```

### Value Descriptions

vdvrOpCodeSetBaudRate	Sets the baud rate.
vdvrOpCodeSetSettingsFlags	Sets the data transmission options. The bit flags are described in <a href="#">Serial Settings Constants</a> .
vdrvOpCodeSetCtsTimeout	Hardware handshake timeout.
vdvrOpCodeClearErr	Clears the hardware error state.
vdvrOpCodeSetSleepMode	Puts the port in sleep mode (not typically used for virtual drivers).

## **Virtual Drivers**

### *Driver Data Structures*

---

vdvrOpCodeSetWakeupMode	Wakes up the port from sleep mode (not typically used for virtual drivers).
vdvrOpCodeFIFOCount	Returns the number of bytes currently in the FIFO (or best estimate).
vdvrOpCodeStartBreak	Sends a break character or enables the sending of break characters.
vdvrOpCodeStopBreak	Stops sending break characters.
vdvrOpCodeStartLoopback	Starts loopback mode (not typically used for virtual drivers).
vdvrOpCodeStopLoopback	Stops loopback mode (not typically used for virtual drivers).
vdrvOpCodeFlushTxFIFO	Flushes the contents of the transmit FIFO.
vdrvOpCodeFlushRxFIFO	Flushes the contents of the receive FIFO.
vdrvOpCodeSendBufferData	Notifies virtual device to send any buffered data it has not emptied from its internal buffers.
vdrvOpCodeRcvCheckIdle	Called periodically to allow the virtual device time to check if there is data to be received. Because virtual devices execute in the same thread as applications, they can be prevented from handling notifications of received data.
vdrvOpCodeEmuSetBlockingHook	Special op code for the Simulator.
vdrvOpCodeGetOptTransmitSize	Returns the optimum buffer size for sending data or returns 0 to specify any buffer size is acceptable.
vdrvOpCodeGetMaxRcvBlockSize	Returns the maximum receive block size that the Serial Manager should request from the virtual device. Can be used to implement flow control.

vdrvOpCodeNotifyBytesReadFromQ	Tells the virtual device that some number of bytes have been read from the receive queue by the client application. Can be used to implement flow control.
vdrvOpCodeSetDTRAsserted	Asserts or de-asserts the DTR signal.
vdrvOpCodeGetDTRAsserted	Gets the status of the DTR signal.
vdrvOpCodeWaitForConfiguration	Waits for USB enumeration to complete. Called from the send and receive functions of the Serial Manager. The driver should have a timeout for how long it waits for enumeration to complete. The driver should return with no error if enumeration has already occurred or has occurred within the driver's timeout. If the enumeration has not occurred within the driver's timeout, the driver should return <code>serErrTimeOut</code> .
vdrvOpCodeGetUSBDeviceDescriptor	Retrieves the device descriptor of a USB driver. Used to gather information about the device's capabilities. Implementation of this op code is optional. If the driver chooses to implement this op code, then the driver should return a pointer to the device descriptor. A driver that chooses not to implement this op code should return <code>serErrNotSupported</code> .
vdrvOpCodeGetUSBConfigDescriptor	Retrieves the configuration descriptor of a USB driver. Used to gather information about the device's capabilities. Implementation of this op code is optional. If the driver chooses to implement this op code, then the driver should return a pointer to the device descriptor. A driver that chooses not to implement this op code should return <code>serErrNotSupported</code> .
vdrvOpCodeEnableIRDA	Enable the IrDA mode and power up the IR line drivers.

## **Virtual Drivers**

### *Driver Data Structures*

---

vdrvOpCodeDisableIRDA	Disable the IrDA mode and disable the IR line drivers.
vdrvOpCodeEnableUART	Powers up the UART and the line drivers.
vdrvOpCodeDisableUART	Powers down the UART and the line drivers.
vdrvOpCodeRxEnable	Enables the receive FIFO, enables UART interrupts, and does whatever else is necessary to allow the UART to receive data.
vdrvOpCodeRxDisable	Disables the receive FIFO and UART interrupts and does whatever is needed to prevent the UART from receiving data.
vdrvOpCodeLineEnable	Enables the main serial line driver for the UART.
vdrvOpCodeEnableUARTInterrupts	Enables the appropriate UART receive interrupts.
vdrvOpCodeDisableUARTInterrupts	Disables all UART interrupts.
vdrvOpCodeSetReceiveQueue	This op code is used by the Serial Manager to set the driver's receive queue. This control code is called when a driver that has previously been opened as a background port is opened as a fully open bidirectional port.
vdrvOpCodeSaveState	Invoked when this port is yielded. This is a hook for the driver to save any current state.
vdrvOpCodeRestoreState	Invoked when the foreground port is closed and this port can become the foreground port.
vdrvOpCodeSetYieldPortCallback	Set the function to be called if the Serial Manager attempts to open another port when this one is open. This op code is for system use only.

vdrvOpCodeSetYieldPortRefCon	Data to pass to the yield port callback function. System use only.
vdvrOpCodeUserDef	User defined function invoked through <a href="#">SrmControl</a> .
vdrvOpCodeSystem	Reserves op codes between 0x7000 and 0x8000 for system use.
vdrvOpCodeCustom	Reserves op codes greater than 0x8000 for driver-specific use.

**Compatibility** The op codes starting at vdrvOpCodeWaitForConfiguration are defined only if [New Serial Manager Feature Set Version 2](#) is present. The op codes for yieldable ports and custom operations are defined only if both [4.0 New Feature Set](#) is present as well.

## Driver Constants

### Driver Version Constants

The driver version constants specify which version of the driver API is implemented by this driver. The [DrvEntryPointProcPtr](#) function passes this value back to the Serial Manager in the drvrVersion field of the [DrvInfoType](#) function.

Constant	Value	Description
kDrvrVersion	4	The latest version of the API.
kDrvrVersion3	3	The version of the driver API that corresponds to <a href="#">New Serial Manager Feature Set Version 1</a> (which ships with roughly Palm OS® 3.3 up to Palm OS 4.0).
kDrvrVersion4	4	The version of the driver API that corresponds to <a href="#">New Serial Manager Feature Set Version 2</a> (which ships with roughly Palm OS 4.0 and higher).

## **Virtual Drivers**

### *Driver Constants*

---

## Port Feature Constants

The port feature constants are flags that describe serial hardware capabilities.

Constant	Value	Description
portPhysicalPort	0x00000001	Should be set for a physical port, unset for a virtual port.
portBkgndModeSupported	0x00000002	Denotes that this port can be used for background ports. This flag is only applicable to virtual drivers. Background mode support is implied on physical drivers.
portRS232Capable	0x00000004	Set if this hardware has an RS-232 port.
portIRDACapable	0x00000008	Set if this hardware has an IR port and supports IrDA mode.
portCradlePort	0x00000010	Set if this hardware controls the cradle port.
portExternalPort	0x00000020	Set if this hardware port is external or on a memory card.
portModemPort	0x00000040	Set if this hardware communicates with a modem.
portCncMgrVisible	0x00000080	Set if this serial port's name is to be displayed in the Connection panel.
portConsolePort	0x00000100	Denotes this hardware controls the console port.
portUSBCapable	0x00000200	Set if this hardware has a USB port.
portPrivateUse	0x00001000	Set if this driver is for special software and not general applications.

**Compatibility**    USB support is only available if [New Serial Manager Feature Set Version 2](#) is present.

## **Virtual Drivers**

### *Virtual Driver-Defined Functions*

---

## **Virtual Driver-Defined Functions**

The functions in this section must be defined by your virtual driver.

### **DrvEntryPointProcPtr**

**Purpose** Entry point for the virtual driver.

**Declared In** SerialDrvr.h

**Prototype** Err (\*DrvEntryPointProcPtr)  
(DrvrEntryOpCodeEnum opCode, void \*uartData)

**Parameters** -> opCode      Entry function code.

<-> uartData      Pointer to data specific to opCode.

**Result** errNone      No error.

-1      The op code is invalid or the hardware could not be found.

**Comments** This function's purpose is based on the value of the opCode parameter. The three possible codes are drvrEntryGetUartFeatures, drvrEntryGetDrvrFuncts, and drvrEntryGetUartFtrsNEntries.

DrvEntryPoint is called with the drvrEntryGetUartFeatures code when the Serial Manager is installed into the system at boot time and is looking for all installed drivers. When this op code is set, the uartData pointer points to a [DrvrInfoType](#) structure. This function does not allocate the structure, it just fills in the fields with information.

This function should check to make sure the associated serial device can operate under the current OS and system settings. If the hardware cannot be found, the function should leave the DrvrInfoType struct untouched and return a -1 error.

The driver needs to supply a string that describes the port it manages. This string is displayed to the user in the Connection panel and is returned by the [SrmGetDeviceInfo](#) function. To set

this string, copy it into the portDesc field of the DrvrInfoType structure. Alternatively, you can supply this string in a driver resource of type 'tSTR' and id kPortDescStrID.

If the DrvrInfoType structure has a positive value in the multipleEnumerations field upon return, the Serial Manager defines one port for each entry in the driver table. The DrvEntryPoint function is called again, this time with the drvrEntryGetUartFtrsNEntries code. The uartData pointer points to a new DrvrInfoType structure whose multipleEnumerations field indicates which port is to be defined. The function should supply all information specific to this port.

DrvEntryPoint is called with the drvrEntryGetDrvrFuncts code when a virtual port is opened. The uartData pointer points to a [VdrvAPIType](#) structure and DrvEntryPoint must fill in the fields of this structure with appropriate function pointers.

<b>Compatibility</b>	Implemented only if <a href="#">New Serial Manager Feature Set Version 1</a> is present.  The drvrEntryGetUartFtrsNEntries is only supported if <a href="#">New Serial Manager Feature Set Version 2</a> is present. This function is fully backwards compatible. Passing 0 for the multipleEnumerations field defines a single port for the driver.
----------------------	--

## VdrvCloseProcPtr

<b>Purpose</b>	Handles all activities needed to close the virtual device.	
<b>Declared In</b>	SerialDrvrv.h	
<b>Prototype</b>	Err (*VdrvCloseProcPtr) (VdrvDataPtr drvrData)	
<b>Parameters</b>	-> drvrData	Pointer to the driver's private global area.
<b>Result</b>	errNone	No error.

## Virtual Drivers

### Virtual Driver-Defined Functions

---

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

## VdrvControlProcPtr

**Purpose** Extends the SrmControl function to the level of the virtual device.

**Declared In** SerialDrvrv.h

**Prototype** Err (\*VdrvControlProcPtr) (VdrvDataPtr drvrData,  
VdrvCtlOpCodeEnum controlCode, void \*controlData,  
UInt16 \*controlDataLen)

**Parameters**

-> drvrData	Pointer to the driver's private global area.
-> controlCode	Control function op code. One of the op codes listed in the <a href="#">VdrvCtlOpCodeEnum</a> type.
<-> controlData	Pointer to data for the specified control function.
<-> controlDataLen	Pointer to length of control data being passed in or out.

**Result**

errNone	No error.
serErrNotSupported	controlCode not supported.
serErrBadParam	controlData or controlDataLen is bad.

**Comments** This function should support the op codes listed in the [VdrvCtlOpCodeEnum](#) type. If this function does not support an op code, it must return the serErrNotSupported error code for that op code.

[Table 64.1](#) shows what is passed for the controlData and controlDataLen parameters for each of the control codes that use them. Control codes not listed do not use these parameters.

**Table 64.1 VDrvControlProcPtr Parameters**

vdvrOpCodeSetBaudRate	-> controlData = Pointer to Int32 (baud rate), -> controlDataLen = Pointer to sizeof(Int32).
vdvrOpCodeSetSettingsFlags	-> controlData = Pointer to UInt32 (bitfield; see <a href="#">Serial Settings Constants</a> ) -> controlDataLen = Pointer to sizeof(UInt32)
vdvrOpCodeFIFOCount	-> controlData = Pointer to Int16, which contains the number of bytes in the FIFO. -> controlDataLen = Pointer to sizeof(Int16).
vdrvOpCodeGetOptTransmitSize	<- controlData = Pointer to Int32 (buffer size), <- controlDataLen = Pointer to sizeof(Int32). Return the optimum buffer size for sending data, or 0 to specify any buffer size is acceptable.
vdrvOpCodeGetMaxRcvBlockSize	<- controlData = Pointer to Int32 (block size), <- controlDataLen = Pointer to sizeof(Int32). Return the maximum block size that the Serial Manager should request from the virtual device.
vdrvOpCodeNotifyBytesReadFromQ	-> controlData = Pointer to Int32 (number of bytes read), -> controlDataLen = Pointer to sizeof(Int32).
vdrvOpCodeSetDTRAsserted	-> controlData = Pointer to Boolean indicating whether to enable or disable DTR. -> controlDataLen = Pointer to sizeof(Boolean)

## **Virtual Drivers**

### *Virtual Driver-Defined Functions*

---

**Table 64.1 VDrvControlProcPtr Parameters (*continued*)**

vdrvOpCodeGetDTRAsserted	<- controlData = Pointer to Boolean indicating whether DTR is enabled. <- controlDataLen = Pointer to sizeof(Boolean)
vdvrOpCodeUserDef	<-> controlData = Pointer from <a href="#">SrmControl</a> (user-defined data), <-> controlDataLen = Pointer to sizeof(Int32).

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

## **VdrvControlCustomProcPtr**

**Purpose** Extends the SrmCustomControl function to the level of the virtual device.

**Declared In** SerialDrvr.h

**Prototype** Err (\*VdrvControlCustomProcPtr)  
(VdrvDataPtr drvrData, UInt16 opCode,  
UInt32 creator, void \*controlData,  
void \*controlDataLenP)

**Parameters**

-> drvrData	Pointer to the driver's private global area.
-> controlCode	Control function op code.
-> creator	Creator ID of the driver that defines the op code. The combination of creator ID and op code uniquely identifies the operation to be performed.
<-> controlData	Pointer to data for the specified control function.

<-> controlDataLen  
Pointer to length of control data being passed  
in or out.

<b>Result</b>	errNone	No error.
	serErrNotSupported	controlCode not supported.
	serErrBadParam	controlData or controlDataLen is bad.

**Comments** This function is a mechanism for a virtual driver to create control codes specific to that driver, allowing for the support of new technologies that have interfaces through the Serial Manager.

**Compatibility** Implemented only if both [New Serial Manager Feature Set Version 2](#) and [4.0 New Feature Set](#) are present.

## VdrvOpenProcPtr

**Purpose** Initializes the virtual device to begin communication.

**Declared In** SerialDrvrv.h

**Prototype** Err (\*VdrvOpenProcPtr) (VdrvDataPtr \*drvrData,  
UIInt32 baudRate, DrvrHWRcvQPtr rcvQP)

<b>Parameters</b>	<-> drvrData	Pointer to a pointer to the driver's private global area (allocated by this function). A pointer to this private global area is passed to the other virtual driver functions.
	-> baudRate	Initial baud rate setting.
	-> rcvQP	Pointer to the driver's receive queue buffer structure. For details on the fields, see <a href="#">DrvrRcvQType</a> .

**Result** errNone No error.

## Virtual Drivers

### Virtual Driver-Defined Functions

---

<b>Comments</b>	This function must allocate and initialize any global variables (and pass back a pointer to a pointer to them in drvrDataP), do any set-up necessary for communicating with other software, and save the rcvQP pointer since it will need the functions and pointers to structures enclosed within to be able to save received data into the Serial Manager's receive queue.
<b>Compatibility</b>	Implemented only if <a href="#">New Serial Manager Feature Set Version 1</a> is present.

## VdrvOpenProcV4Ptr

<b>Purpose</b>	Initializes the virtual device to begin communication.	
<b>Declared In</b>	SerialDrvrv.h	
<b>Prototype</b>	Err (*VdrvOpenProcV4Ptr) (VdrvDataPtr *drvrData, VdrvConfigPtr configP, DrvrHWRcvQPtr rcvQP)	
<b>Parameters</b>	<-> drvrData	Pointer to a pointer to the driver's private global area (allocated by this function). A pointer to this private global area is passed to the other virtual driver functions.
	-> configP	Pointer to the configuration structure specifying the port's properties. See <a href="#">VdrvConfigType</a> .
	-> rcvQP	Pointer to the driver's receive queue buffer structure. For details on the fields, see <a href="#">DrvrRcvQType</a> .
<b>Result</b>	errNone	No error.
<b>Comments</b>	This function must allocate and initialize any global variables (and pass back a pointer to a pointer to them in drvrDataP), do any set-up necessary for communicating with other software, and save the rcvQP pointer since it will need the functions and pointers to structures enclosed within to be able to save received data into the Serial Manager's receive queue.	

**Compatibility** Implemented only if both [New Serial Manager Feature Set Version 2](#) and [4.0 New Feature Set](#) are present.

## VdrvStatusProcPtr

**Purpose** Returns virtual device status.

**Declared In** SerialDrvrv.h

**Prototype** UInt16 (\*VDrvStatusProcPtr) (VdrvDataPtr drvrData)

**Parameters** -> drvrData Pointer to the driver's private global area.

**Result** An unsigned long bitfield denoting the status of the virtual device, but only if the virtual device is emulating hardware. The individual bit flags are described in the [DrvrvStatusEnum](#) type.

**Comments** Generally, status is returned only to the client application using the virtual device. The Serial Manager does not use status information from virtual devices.

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

## VdrvWriteProcPtr

**Purpose** Writes a block of bytes.

**Declared In** SerialDrvrv.h

**Prototype** UInt32 (\*VdrvWriteProcPtr) (VdrvDataPtr drvrDataP, void \*bufP, UInt32 size, Err \*errP)

**Parameters** -> drvrDataP Pointer to the driver's private global area.  
-> bufP Pointer to buffer containing the data to be written to the virtual device.  
-> size Number of bytes in the buffer bufP.

## **Virtual Drivers**

### *Serial Manager Queue Functions*

---

<- errP                    Pointer to an error code resulting from the operation. Zero is returned if there is no error.

**Result**    Returns the actual number of bytes written.

**Compatibility**    Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

## **Serial Manager Queue Functions**

The functions in this section are supplied by the Serial Manager to the virtual driver through the [DrvrRcvQType](#) passed to the [VdrvOpenProcPtr](#) function.

### **GetSizeProcPtr**

**Purpose**    Returns the total size of the Serial Manager's receive queue.

**Declared In**    `SerialDrvrv.h`

**Prototype**    `typedef UInt32 (*GetSizeProcPtr) (void *theQ)`

**Parameters**    `-> theQ`                    Pointer to the receive queue.

**Result**    Size in bytes of the Serial Manager's receive queue.

**Comments**    This function is useful for implementing flow control.

**Compatibility**    Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

## GetSpaceProcPtr

<b>Purpose</b>	Returns the available space in the Serial Manager's receive queue.
<b>Declared In</b>	SerialDrvr.h
<b>Prototype</b>	<code>typedef UInt32 (*GetSpaceProcPtr) (void *theQ)</code>
<b>Parameters</b>	<code>-&gt; theQ</code> Pointer to the receive queue.
<b>Result</b>	Size in bytes of the available space in the Serial Manager's receive queue.
<b>Comments</b>	This function is useful for implementing flow control.
<b>Compatibility</b>	Implemented only if <a href="#">New Serial Manager Feature Set Version 1</a> is present.

## SignalCheckPtr

<b>Purpose</b>	Check the queue to see if the semaphore needs to be signalled.
<b>Declared In</b>	SerialDrvr.h
<b>Prototype</b>	<code>typedef void (*SignalCheckPtr) (void *theQ, UInt16 lineErrsP)</code>
<b>Parameters</b>	<code>-&gt; theQ</code> Pointer to the receive queue. <code>-&gt; lineErrsP</code> Any serial line errors received should be reported here.
<b>Result</b>	Returns nothing.
<b>Comments</b>	This function signals that there is data to be received without writing anything to the receive queue. The <a href="#">WriteByteProcPtr</a> and <a href="#">WriteBlockProcPtr</a> functions also signal that there is data to be received after they have written the data to the queue.

## Virtual Drivers

### Serial Manager Queue Functions

---

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 2](#) is present.

## WriteBlockProcPtr

**Purpose** Writes a block of bytes to the Serial Manager's receive queue.

**Declared In** SerialDrvrvr.h

**Prototype** `typedef Err (*WriteBlockProcPtr) (void *theQ,  
                          UInt8 *bufP, UInt16 size, UInt16 lineErrs)`

**Parameters**

-> theQ	Pointer to the receive queue.
-> bufP	Pointer to the buffer holding bytes to be written.
-> size	Size of bufP.
-> lineErrs	Any serial line errors received should be reported here.

**Result**

errNone	No error.
serErrLineErr	There was a software overrun line error.

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

## WriteByteProcPtr

**Purpose** Writes one byte to the Serial Manager's receive queue.

**Declared In** SerialDrvrvr.h

**Prototype** `typedef Err (*WriteByteProcPtr) (void *theQ,  
                          UInt8 theByte, UInt16 lineErrs)`

**Parameters**

-> theQ	Pointer to the receive queue.
-> theByte	The byte to be written to the queue.

	-> lineErrs	Any serial line errors received should be reported here.
<b>Result</b>	errNone	No error.
	serErrLineErr	There was a software overrun line error.

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

## **Virtual Drivers**

### *Serial Manager Queue Functions*

---

# Serial Manager

---

This chapter provides reference material for the Serial Manager API:

- [Serial Manager Data Structures](#)
- [Serial Manager Constants](#)
- [Serial Manager Functions](#)
- [Serial Manager Application-Defined Functions](#)

The header file `SerialMgr.h` declares the Serial Manager API. The file `SystemResources.h` defines some serial port constants. For more information on the Serial Manager, see the chapter “[Serial Communication](#)” on page 89 of the *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## Serial Manager Data Structures

### **DeviceInfoType**

The `DeviceInfoType` structure defines information about a serial device. This structure is returned by the [`SrmGetDeviceInfo`](#) function.

```
typedef struct DeviceInfoType {  
    UInt32 serDevCreator;  
    UInt32 serDevFtrInfo;  
    UInt32 serDevMaxBaudRate;  
    UInt32 serDevHandshakeBaud;  
    Char *serDevPortInfoStr;  
    UInt8 reserved[8];  
} DeviceInfoType;  
typedef DeviceInfoType *DeviceInfoPtr;
```

## Serial Manager

### *Serial Manager Data Structures*

---

#### Value Descriptions

serDevCreator	Four-character creator ID for serial driver.
serDevFtrInfo	Flags defining features of this serial hardware. See <a href="#">Serial Capabilities Constants</a> for a description of these flags.
serDevMaxBaudRate	Maximum baud rate for this device.
serDevHandshakeBaud	Hardware handshaking is recommended for baud rates over this rate.
serDevPortInfoStr	Description of serial hardware device or virtual device.

#### SrmCtlEnum

The SrmCtlEnum enumerated type specifies a serial control operation. Specify one of these enumerated types for the op parameter to the [SrmControl](#) call.

```
typedef enum SrmCtlEnum {
    srmCtlFirstReserved = 0,
    srmCtlSetBaudRate,
    srmCtlGetBaudRate,
    srmCtlSetFlags,
    srmCtlGetFlags,
    srmCtlSetCtsTimeout,
    srmCtlGetCtsTimeout,
    srmCtlStartBreak,
    srmCtlStopBreak,
    srmCtlStartLocalLoopback,
    srmCtlStopLocalLoopback,
    srmCtlIrDAEnable,
    srmCtlIrDADisable,
    srmCtlRxEnable,
    srmCtlRxDisable,
    srmCtlEmuSetBlockingHook,
    srmCtlUserDef,
```

```
srmCtlGetOptimalTransmitSize,  
srmCtlSetDTRAsserted,  
srmCtlGetDTRAsserted,  
srmCtlSetYieldPortCallback,  
srmCtlSetYieldPortRefCon,  
srmCtlSystemReserved = 0x7000  
srmCtlCustom = 0x8000,  
srmCtlLAST  
} SrmCtlEnum;
```

### Value Descriptions

srmCtlSetBaudRate	Sets the current baud rate for the serial hardware.
srmCtlGetBaudRate	Gets the current baud rate for the serial hardware.
srmCtlSetFlags	Sets the current flag settings for the serial hardware. Specify flags from the set described in <a href="#">Serial Settings Constants</a> .
srmCtlGetFlags	Gets the current flag settings for the serial hardware.
srmCtlSetCtsTimeout	Sets the current CTS timeout value for hardware handshaking.
srmCtlGetCtsTimeout	Gets the current CTS timeout value for hardware handshaking.
srmCtlStartBreak	Turn RS-232 break signal on. Caller is responsible for turning this signal on and off and insuring it is on long enough to generate a viable break.
srmCtlStopBreak	Turn RS-232 break signal off.
srmCtlStartLocalLoopback	Start local loopback test.
srmCtlStopLocalLoopback	Stop local loopback test.

## Serial Manager

### *Serial Manager Data Structures*

---

srmCtlIrDAEnable	Enable IrDA connection on this serial port.
	<b>NOTE:</b> You cannot enable an IrDA connection on a VZ processor.
srmCtlIrDADisable	Disable IrDA connection on this serial port.
srmCtlRxEnable	Enable receiver (for IrDA).
srmCtlRxDisable	Disable receiver (for IrDA).
srmCtlEmuSetBlockingHook	Set a blocking hook routine for emulation mode only. Not supported on the actual device.
srmCtlUserDef	This is a user-defined function that third-party hardware developers can use to set or retrieve hardware-specific information from the serial driver. This op code invokes the driver's corresponding control function with its user-defined op code and the parameters are passed directly through to the serial driver. A serial driver that does not handle this function returns a <code>serErrBadParam</code> error.  The <code>srmCtlUserDef</code> op code is superseded by defining a custom op code if <a href="#">New Serial Manager Feature Set Version 2</a> is present.
srmCtlGetOptimalTransmitSize	Ask the port for the most efficient buffer size for transmitting data packets. This op code returns an error (buffering not necessary), 0 (buffering requested, but application can choose buffer size), or a number greater than 0 (recommended buffer size).
srmCtlSetDTRAsserted	Enabled or disable the DTR signal. This is not supported by all hardware.
srmCtlGetDTRAsserted	Ask the port whether the DTR signal is enabled or disabled.

srmCtlSetYieldPortCallback	Set the function to be called if the Serial Manager attempts to open another port when this one is open. This op code is for system use only.
srmCtlSetYieldPortRefCon	Data to pass to the yield port callback function. System use only.
srmCtlSystemReserved	Reserves op codes between 0x7000 and 0x8000 for system use.
srmCtlCustom	Reserves op codes greater than 0x8000 for driver-specific use.

**Compatibility** Custom control op codes are only supported if both [New Serial Manager Feature Set Version 2](#) and [4.0 New Feature Set](#) are present.

## SrmOpenConfigType

The SrmOpenConfigType structure specifies parameters for opening a serial port. This structure is passed as a parameter to [SrmExtOpen](#).

```
typedef struct SrmOpenConfigType {  
    UInt32 baud;  
    UInt32 function;  
    MemPtr drvrDataP;  
    UInt16 drvrDataSize;  
    UInt32 sysReserved1;  
    UInt32 sysReserved2;  
} SrmOpenConfigType;
```

## **Serial Manager**

### *Serial Manager Data Structures*

---

#### **Field Descriptions**

baud	Baud rate at which to open the connection. Serial drivers that do not require baud rates ignore this field.
function	The reason why the port was opened. Specify the creator ID of the application that is opening the port or one of the following values:  <code>serFncUndefined</code> Undefined function. This is the default value for this field.  <code>serFncPPPSession</code> The connection is to be used for the PPP interface.  <code>serFncSLIPSession</code> The connection is to be used for the SLIP session.  <code>serFncDebugger</code> The connection is to be used for a debugging session.  <code>serFncHotSync</code> The connection is to be used for a HotSync operation.  <code>serFncConsole</code> The connection is to the debugging console.  <code>serFncTelephony</code> The connection is to the telephony library.
	The function field is used by protocols such as USB and Bluetooth that perform different setup tasks based on which type of application is using them. RS-232 drivers ignore this parameter.
<code>drvrvDataP</code>	Pointer to a driver-specific data block.
<code>drvrvDataSize</code>	The size of the data block pointed to by <code>drvrvDataP</code> .

sysReserved1 Reserved for future use.

sysReserved2 Reserved for future use.

**Compatibility** This structure is only defined if both [New Serial Manager Feature Set Version 2](#) and [4.0 New Feature Set](#) are present.

## Serial Manager Constants

### Port Constants

When you specify the port to open in the [SrmOpen](#), [SrmOpenBackground](#), [SrmExtOpen](#), or [SrmExtOpenBackground](#) call, you can use either a logical port constant, physical port constant, or a virtual port constant, but it is highly recommended that you use a logical port constant wherever possible.

#### Logical Serial Port Constants

These constants specify the logical port names.

Constant	Value	Description
serPortLocalHotSync	0x8000	The physical HotSync port. The Serial Manager automatically detects whether this port is USB or RS-232.
serPortCradlePort	0x8000	Cradle port. The Serial Manager automatically detects whether this port is USB or RS-232. Most applications should specify this as the port.
serPortIrPort	0x8001	The IR port. This is a raw IrDA port with no protocol support.
serPortConsolePort	0x8002	The debug console port, either USB or RS-232. USB is preferred where both are available.

## Serial Manager

### Serial Manager Constants

---

Constant	Value	Description
serPortCradleRS232Port	0x8003	Port for the RS-232 cradle. Specify this port if you want to ensure that your application uses RS-232 communications only.
serPortCradleUSBPort	0x8004	Port for the USB cradle. Specify this port if you want to ensure that your application uses USB communications only.

**Compatibility** USB ports are only supported if [New Serial Manager Feature Set Version 2](#) is present.

### Physical Serial Port Constants

The physical port constants specify 4-character constants that reference the physical hardware of the device. Doing so is **not** recommended because the hardware they reference may not exist on a particular device.

Physical port	Value	Description
sysFileCUart328	'u328'	Cradle port using the 68328 UART. This port can be switched between RS232 and IrDA mode using the <a href="#">SrmControl</a> call.
sysFileCUart328EZ	'u8EZ'	Cradle port using the 68328EZ UART. This port can also be switched between RS232 and IrDA mode.
sysFileCUart650	'u650'	Specifies the IR port on the upgrade card for Palm Personal or Palm Professional devices. This gives you a raw IR port like calling <a href="#">SrmControl</a> does, but it only exists on devices that have the upgrade card.

### Virtual Serial Port Constants

The virtual port constants specify 4-character constants that identify virtual ports, simulating a hardware interface. Virtual ports are not tied to specific hardware.

<b>Physical port</b>	<b>Value</b>	<b>Description</b>
sysFileCVirtIrComm	'ircm'	A virtual serial cable over an IrDA link using the IRComm protocol. It can only be used to talk to another IRComm device.
sysFileCVirtRfComm	'rfcm'	RFCOMM (Bluetooth) virtual port plug-in.
sysFileCBtConnectPanelHelper	'btcp'	Bluetooth Connection Panel helper application.

**Compatibility** All virtual port constants other than `sysFileCVirtIrComm` are only defined if both [New Serial Manager Feature Set Version 2](#) and [4.0 New Feature Set](#) are present.

## Serial Capabilities Constants

The serial capabilities constant flags describe serial hardware capabilities. These flags are set in the `serDevFtrInfo` field of the [`DeviceInfoType`](#) structure.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
serDevCradlePort	0x00000001	Serial hardware controls RS-232 serial from cradle connector of Palm device.
serDevRS232Serial	0x00000002	Serial hardware has RS-232 line drivers.
serDevIRDACapable	0x00000004	Serial hardware has IR line drivers and generates IrDA mode serial signals.
serDevModemPort	0x00000008	Serial hardware drives modem connection.
serDevCncMgrVisible	0x00000010	Serial device port name string is to be displayed in the Connection panel.

## Serial Manager

### *Serial Manager Constants*

---

Constant	Value	Description
serDevConsolePort	0x00000020	Serial device is the default console port.
serDevUSBCapable	0x00000040	Serial hardware controls USB serial from cradle connector of Palm device.

**Compatibility** USB ports are only supported if [New Serial Manager Feature Set Version 2](#) is present.

## Serial Settings Constants

The serial settings constants identify bit flags that correspond to various serial hardware settings. Use [SrmControl](#) with the op code srmCtlSetFlags to control which settings are used.

Constant	Value	Description
srmSettingsFlagStopBitsM	0x00000001	Mask for stop bits field
srmSettingsFlagStopBits1	0x00000000	1 stop bit (default)
srmSettingsFlagStopBits2	0x00000001	2 stop bits
srmSettingsFlagParityOnM	0x00000002	Mask for parity on
srmSettingsFlagParityEvenM	0x00000004	Mask for parity even
srmSettingsFlagXonXoffM	0x00000008	Mask for Xon/Xoff flow control (not implemented)
srmSettingsFlagRTSAutoM	0x00000010	Mask for RTS receive flow control. This is the default.
srmSettingsFlagCTSAutoM	0x00000020	Mask for CTS transmit flow control
srmSettingsFlagBitsPerCharM	0x000000C0	Mask for bits per character
srmSettingsFlagBitsPerChar5	0x00000000	5 bits per character
srmSettingsFlagBitsPerChar6	0x00000040	6 bits per character

## Serial Manager

### *Serial Manager Constants*

---

<b>Constant</b>	<b>Value</b>	<b>Description</b>
srmSettingsFlagBitsPerChar7	0x00000080	7 bits per character
srmSettingsFlagBitsPerChar8	0x000000C0	8 bits per character (default)
srmSettingsFlagFlowControlIn	0x00000100	Protect the receive buffer from software overruns. When this flag and srmSettingsFlagRTSAutoM are set, which is the default case, it causes the Serial Manager to assert RTS to prevent the transmitting device from continuing to send data when the receive buffer is full. Once the application receives data from the buffer, RTS is de-asserted to allow data reception to resume.  Note that this feature effectively prevents software overrun line errors but may also cause CTS timeouts on the transmitting device if the RTS line is asserted longer than the defined CTS timeout value.
srmSettingsFlagRTSInactive	0x00000200	If this flag is set and srmSettingsFlagRTSAutoM is not set, RTS is held in the inactive (flow off) state forever.

---

## **Serial Manager**

### *Serial Manager Constants*

---

## **Status Constants**

The status constants identify bit flags that correspond to the status of serial signals. They can be returned by the [SrmGetStatus](#) function.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
srmStatusCtsOn	0x00000001	CTS line is active.
srmStatusRtsOn	0x00000002	RTS line is active.
srmStatusDsrOn	0x00000004	DSR line is active.
srmStatusBreakSigOn	0x00000008	Break signal is active.

## **Line Error Constants**

The line error constants identify bit flags that correspond to the line errors that may occur on the port. They can be returned by the [SrmGetStatus](#) function.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
serLineErrorParity	0x0001	Parity error
serLineErrorHWOVERRUN	0x0002	Hardware overrun
serLineErrorFraming	0x0004	Framing error
serLineErrorBreak	0x0008	Break signal asserted
serLineErrorHShake	0x0010	Line handshake error
serLineErrorSWOVERRUN	0x0020	Software overrun
serLineErrorCarrierLost	0x0040	Carrier detect signal dropped

# Serial Manager Functions

## SrmClearErr

**Purpose** Clears the port of any line errors.

**Declared In** SerialMgr.h

**Prototype** Err SrmClearErr (UInt16 portId)

**Parameters** -> portID Port ID returned from [SrmOpen](#) or [SrmExtOpen](#).

**Result** This function returns the following error codes:

errNone No error.

serErrNotSupported  
The port is not the foreground port.

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

## SrmClose

**Purpose** Closes a serial port and makes it available to other applications, regardless of whether the port is a foreground or background port.

**Declared In** SerialMgr.h

**Prototype** Err SrmClose (UInt16 portId)

**Parameters** -> portID Port ID for port to be closed.

**Result** This function returns the following error codes:

errNone No error.

serErrBadPort This port doesn't exist.

## Serial Manager

### Serial Manager Functions

---

serErrNotOpen The serial port is not open.

serErrNoDevicesAvail  
No serial devices could be found.

<b>Comments</b>	If a foreground port is being closed and a background port exists, the background will have access to the port as long as another foreground port is not opened.  If a foreground port is being closed and a yielded port exists, the yielded port will have access to the port as long as it does not yield to the opening of another foreground port. If there are both a yielded port and a background port for the foreground port being closed, the yielded port takes precedence over the background port.
-----------------	--

<b>Compatibility</b>	Implemented only if <a href="#">New Serial Manager Feature Set Version 1</a> is present.
----------------------	--

<b>See Also</b>	<a href="#">SrmOpen</a> , <a href="#">SrmOpenBackground</a>
-----------------	---

## SrmControl

<b>Purpose</b>	Performs a serial control function.
----------------	-------------------------------------

<b>Declared In</b>	SerialMgr.h
--------------------	-------------

<b>Prototype</b>	Err SrmControl (UInt16 portID, UInt16 op, void *valueP, UInt16 *valueLenP)
------------------	---

<b>Parameters</b>	-> portID	Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a> .
	-> op	Control operation to perform. Specify one of the <a href="#">SrmCtlEnum</a> enumerated types.
	<-> valueP	Pointer to a value to use for the operation. See Comments for details.
	<-> valueLenP	Pointer to the size of *valueP. See Comments for details.

<b>Result</b>	This function returns the following error codes:
---------------	--

errNone	No error.
serErrBadParam	An invalid op code was specified.
serErrBadPort	This port doesn't exist.
serErrNotOpen	The serial port is not open.
serErrNoDevicesAvail	No serial devices could be found.
serErrNotSupported	The specified op code is not supported in the current configuration.

**Comments** [Table 65.1](#) shows what to pass for the `valueP` and `valueLenP` parameters for each of the operation codes. Control codes not listed do not use these parameters. See [SrmCtlEnum](#) for a complete list of control codes.

**Table 65.1 SrmControl Parameters**

Operation Code	Parameters
<code>srmCtlSetBaudRate</code>	-> <code>valueP</code> = Pointer to <code>Int32</code> (baud rate) -> <code>valueLenP</code> = Pointer to <code>sizeof(Int32)</code>
<code>srmCtlGetBaudRate</code>	<- <code>valueP</code> = Pointer to <code>Int32</code> (baud rate) <- <code>valueLenP</code> = Pointer to <code>Int16</code>
<code>srmCtlSetFlags</code>	-> <code>valueP</code> = Pointer to <code>UInt32</code> (bitfield; see <a href="#">Serial Settings Constants</a> ) -> <code>valueLenP</code> = Pointer to <code>sizeof(UInt32)</code>
<code>srmCtlGetFlags</code>	<- <code>valueP</code> = Pointer to <code>UInt32</code> (bitfield) <- <code>valueLenP</code> = Pointer to <code>Int16</code>
<code>srmCtlSetCtsTimeout</code>	-> <code>valueP</code> = Pointer to <code>Int32</code> (timeout value) -> <code>valueLenP</code> = Pointer to <code>sizeof(Int32)</code>
<code>srmCtlGetCtsTimeout</code>	<- <code>valueP</code> = Pointer to <code>Int32</code> (timeout value) <- <code>valueLenP</code> = Pointer to <code>Int16</code>

## **Serial Manager**

### *Serial Manager Functions*

---

**Table 65.1 SrmControl Parameters (*continued*)**

<b>Operation Code</b>	<b>Parameters</b>
srmCtlUserDef	<-> valueP = Pointer passed to the serial or virtual driver <-> valueLenP = Pointer to sizeof(Int32) For a serial driver, these pointers are passed to the driver's control function and they contain that functions return values (if any) upon return.
srmCtlGetOptimalTransmitSize	<- valueP = Pointer to Int32 <- valueLenP = Pointer to sizeof(Int32) If an error is returned by SrmControl, no buffering should be done. If valueP points to zero, buffering is requested, but the transmitting application cannot determine the buffer size. If valueP points to a number > 0, then try to send data in blocks of this number of bytes, as this is the most efficient block size for this particular device.
srmCtlSetDTRAsserted	-> valueP = Pointer to Boolean indicating whether to enable or disable DTR. -> valueLenP = Pointer to sizeof(Boolean)
srmCtlGetDTRAsserted	<- valueP = Pointer to Boolean indicating whether DTR is enabled. <- valueLenP = Pointer to Int16

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**See Also** [SrmCustomControl](#)

## SrmCustomControl

**Purpose** Performs a custom serial control function.

**Declared In** SerialMgr.h

**Prototype** Err SrmCustomControl (UInt16 portID,  
                  UInt16 opCode, UInt32 creator, void \*valueP,  
                  UInt16 \*valueLenP)

<b>Parameters</b>	-> portID	Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a> .
	-> opCode	Control operation to perform. The op code must be greater than srmCtlCustom.
	-> creator	Creator ID of the driver that defines the op code. The combination of creator ID and op code uniquely identifies the operation to be performed.
	<-> valueP	Pointer to a value to use for the operation.
	<-> valueLenP	Pointer to the size of *valueP.

**Result** This function returns the following error codes:

errNone	No error.
serErrNotSupported	The port is not the foreground port.
serErrBadPort	This port doesn't exist.
serErrNotOpen	The serial port is not open.
serErrNoDevicesAvail	No serial devices could be found.

**Comments** This function is a mechanism for a virtual driver to create control codes specific to that driver, allowing for the support of new technologies that have interfaces through the Serial Manager.  
This function simply forwards the opCode and any valueP parameter to the virtual driver for the port. The virtual driver may

## Serial Manager

### *Serial Manager Functions*

---

return its own error code if the opCode or the input in valueP is invalid.

**Compatibility** Implemented only if both [New Serial Manager Feature Set Version 2](#) and [4.0 New Feature Set](#) are present.

**See Also** [SrmControl](#)

## SrmExtOpen

**Purpose** Opens a foreground port connection with the specified configuration.

**Declared In** SerialMgr.h

**Prototype** Err SrmExtOpen (UInt32 port,  
SrmOpenConfigType \*configP, UInt16 configSize,  
UInt16 \*newPortIdP)

**Parameters** -> port The four-character port name (such as 'ircm' or 'u328') or logical port number to be opened. (See [Port Constants](#).)

-> configP Pointer to the configuration structure specifying the serial port's properties. See [SrmOpenConfigType](#).

-> configSize The size of the configuration structure pointed to by configP.

<- newPortIdP Contains the port ID to be passed to other Serial Manager functions.

**Result** This function returns the following error codes:

errNone No error.

serErrBadPort The port parameter does not specify a valid port.

serErrBadParam The configP parameter is NULL.

serErrAlreadyOpen

The Serial Manager already has a port open.

memErrNotEnoughSpace

There was not enough memory available to open the port.

**Comments**

Do not keep the port open any longer than necessary. An open serial port consumes more energy from the device's batteries.

The values specified in the configP parameter depend on the type of connection being made. For RS-232 connections, you specify the baud rate but not a purpose. For USB connections, you specify a purpose but not a baud rate.

A newly opened port has its line errors cleared, the default CTS timeout set (specified by the constant `srmDefaultCTSTimeout`), a 512-byte receive queue allocated, 1 stop bit, 8 bits per character, RTS enabled, and flow control enabled. To increase the receive queue size, use [SrmSetReceiveBuffer](#). To change the other serial port settings, use [SrmControl](#).

**Compatibility**

Implemented only if both [New Serial Manager Feature Set Version 2](#) and [4.0 New Feature Set](#) are present. The `SrmExtOpen` function replaces the [SrmOpen](#) function.

**See Also**

[SrmOpen](#), [SrmExtOpenBackground](#)

## SrmExtOpenBackground

**Purpose** Opens a port with the specified configuration in the background. Background ports relinquish control when another task opens the port with the [SrmOpen](#) or [SrmExtOpen](#) call.

**Declared In** SerialMgr.h

**Prototype** Err SrmExtOpenBackground (UInt32 port,  
SrmOpenConfigType \*configP, UInt16 configSize,  
UInt16 \*newPortIdP)

**Parameters**

-> port	Physical or logical port number to be opened. See <a href="#">Port Constants</a> for more information.
-> configP	Pointer to the configuration structure specifying the serial port's properties. See <a href="#">SrmOpenConfigType</a> .
-> configSize	The size of the configuration structure pointed to by configP.
<- newPortIdP	Contains the port ID to be passed to other Serial Manager functions.

**Result** This function returns the following error codes:

errNone	No error.
serErrAlreadyOpen	This port already has an installed background owner.
serErrBadPort	This port doesn't exist.
serErrNotSupported	This type of port cannot be opened in the background.
serErrBadParam	The configP parameter is NULL.
memErrNotEnoughSpace	There was not enough memory available to open the port.

<b>Comments</b>	This function is provided to support tasks that want to use a serial device to receive data only when no other task is using the port.  If a background port is forced to surrender control of the hardware as a result of another task opening a foreground connection, all buffers for the background port are flushed. After this active task closes the port, active control of the port is returned to the background task. Only one task can have background ownership of the port.  Note that background ports have limited functionality: they can only receive data and notify owning clients of what data has been received.  The values specified in the configP parameter depend on the type of connection being made. For RS-232 connections, you specify the baud rate but not a purpose. For USB connections, you specify a purpose but not a baud rate.
<b>Compatibility</b>	Implemented only if both <a href="#">New Serial Manager Feature Set Version 2</a> and <a href="#">4.0 New Feature Set</a> are present. The SrmExtOpenBackground function replaces the <a href="#">SrmOpenBackground</a> function.

**See Also** [SrmOpen](#), [SrmExtOpen](#)

## SrmGetDeviceCount

<b>Purpose</b>	Returns the number of available serial devices.	
<b>Declared In</b>	<code>SerialMgr.h</code>	
<b>Prototype</b>	<code>Err SrmGetDeviceCount (UInt16 *numOfDevicesP)</code>	
<b>Parameters</b>	<code>&lt;- numOfDevicesP</code>	Pointer to address where the number of serial devices is returned.
<b>Result</b>	<code>errNone</code>	No error.

## Serial Manager

### Serial Manager Functions

---

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**See Also** [SrmGetDeviceInfo](#)

## SrmGetDeviceInfo

**Purpose** Returns information about a serial device.

**Declared In** SerialMgr.h

**Prototype** Err SrmGetDeviceInfo (UInt32 deviceID,  
DeviceInfoType \*deviceInfoP)

**Parameters** -> deviceID ID of serial device to get information for. You can pass a zero-based index (0, 1, 2, ...), a valid port ID returned from [SrmOpen](#) or [SrmExtOpen](#), or a 4-character port name (such as 'u328', 'u650', or 'ircm'). See [Port Constants](#).  
<- deviceInfoP Pointer to a [DeviceInfoType](#) structure where information about the device is returned.

**Result** This function returns the following error codes:

errNone No error.

serErrBadPort This port doesn't exist.

serErrNoDevicesAvail  
The Serial Manager cannot find any serial devices.

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**See Also** [SrmGetDeviceCount](#)

## SrmGetStatus

<b>Purpose</b>	Returns status information about the serial hardware.						
<b>Declared In</b>	<code>SerialMgr.h</code>						
<b>Prototype</b>	<code>Err SrmGetStatus (UInt16 portId,                   UInt32 *statusFieldP, UInt16 *lineErrsP)</code>						
<b>Parameters</b>	<table><tr><td><code>-&gt; portID</code></td><td>Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a>.</td></tr><tr><td><code>&lt;- statusFieldP</code></td><td>Pointer to address where hardware status information for the port is returned. This is a 32-bit field using the flags described in <a href="#">Status Constants</a>.</td></tr><tr><td><code>&lt;- lineErrsP</code></td><td>Pointer to address where the number of line errors for the port is returned. The line error flags are described in <a href="#">Line Error Constants</a>.</td></tr></table>	<code>-&gt; portID</code>	Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a> .	<code>&lt;- statusFieldP</code>	Pointer to address where hardware status information for the port is returned. This is a 32-bit field using the flags described in <a href="#">Status Constants</a> .	<code>&lt;- lineErrsP</code>	Pointer to address where the number of line errors for the port is returned. The line error flags are described in <a href="#">Line Error Constants</a> .
<code>-&gt; portID</code>	Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a> .						
<code>&lt;- statusFieldP</code>	Pointer to address where hardware status information for the port is returned. This is a 32-bit field using the flags described in <a href="#">Status Constants</a> .						
<code>&lt;- lineErrsP</code>	Pointer to address where the number of line errors for the port is returned. The line error flags are described in <a href="#">Line Error Constants</a> .						
<b>Result</b>	This function returns the following error codes:  <code>errNone</code> No error.  <code>serErrBadPort</code> This port doesn't exist.  <code>serErrNotSupported</code> The port is a yielded port.  <code>serErrNoDevicesAvail</code> No serial devices could be found.						
<b>Comments</b>	Typically, <code>SrmGetStatus</code> is called to retrieve the line errors for the port if some of the send and receive functions return a <code>serErrLineErr</code> error code.						
<b>Compatibility</b>	Implemented only if <a href="#">New Serial Manager Feature Set Version 1</a> is present.						

## Serial Manager

### *Serial Manager Functions*

---

## SrmOpen

<b>Purpose</b>	Opens a foreground port connection with the specified port name or logical port number.								
<b>Declared In</b>	SerialMgr.h								
<b>Prototype</b>	<pre>Err SrmOpen (UInt32 port, UInt32 baud, UInt16 *newPortIdP)</pre>								
<b>Parameters</b>	<table><tr><td>-&gt; port</td><td>The four-character port name or logical port number to be opened. See <a href="#">Port Constants</a> for more information.</td></tr><tr><td>-&gt; baud</td><td>Initial baud rate of port.</td></tr><tr><td>&lt;- newPortIdP</td><td>Contains the port ID to be passed to other Serial Manager functions.</td></tr></table>	-> port	The four-character port name or logical port number to be opened. See <a href="#">Port Constants</a> for more information.	-> baud	Initial baud rate of port.	<- newPortIdP	Contains the port ID to be passed to other Serial Manager functions.		
-> port	The four-character port name or logical port number to be opened. See <a href="#">Port Constants</a> for more information.								
-> baud	Initial baud rate of port.								
<- newPortIdP	Contains the port ID to be passed to other Serial Manager functions.								
<b>Result</b>	This function returns the following error codes:  <table><tr><td>errNone</td><td>No error.</td></tr><tr><td>serErrAlreadyOpen</td><td>This port already has an installed foreground owner.</td></tr><tr><td>serErrBadPort</td><td>This port doesn't exist.</td></tr><tr><td>memErrNotEnoughSpace</td><td>There was not enough memory available to open the port.</td></tr></table>	errNone	No error.	serErrAlreadyOpen	This port already has an installed foreground owner.	serErrBadPort	This port doesn't exist.	memErrNotEnoughSpace	There was not enough memory available to open the port.
errNone	No error.								
serErrAlreadyOpen	This port already has an installed foreground owner.								
serErrBadPort	This port doesn't exist.								
memErrNotEnoughSpace	There was not enough memory available to open the port.								
<b>Comments</b>	Only one application or task may have access to a particular serial port at any time.  Do not keep the port open any longer than necessary. An open serial port consumes more energy from the device's batteries.								
<b>Compatibility</b>	Implemented only if <a href="#">New Serial Manager Feature Set Version 1</a> is present.								

If [New Serial Manager Feature Set Version 2](#) is present, the SrmOpen function is replaced by [SrmExtOpen](#). SrmOpen is supported for backward compatibility.

**See Also** [SrmOpenBackground](#)

## SrmOpenBackground

**Purpose** Allows a task to open, initialize, and use the port, but always relinquishes control of the port when another task opens the port with the [SrmOpen](#) call.

**Declared In** SerialMgr.h

**Prototype** Err SrmOpenBackground (UInt32 port, UInt32 baud,  
UInt16 \*newPortIdP)

**Parameters**

-> port	The four-character port name or logical port number to be opened. See <a href="#">Port Constants</a> for more information.
-> baud	Initial baud rate of port.
<- newPortIdP	Contains the port ID to be passed to other Serial Manager functions.

**Result** This function returns the following error codes:

errNone No error.

serErrAlreadyOpen This port already has an installed background owner.

serErrBadPort This port doesn't exist.

memErrNotEnoughSpace There was not enough memory available to open the port.

**Comments** This function is provided to support tasks that want to use a serial device to receive data only when no other task is using the port.

## Serial Manager

### *Serial Manager Functions*

---

If a background port is forced to surrender control of the hardware as a result of another task opening a foreground connection, all buffers for the background port are flushed. After this active task closes the port, active control of the port is returned to the background task. Only one task can have background ownership of the port.

Note that background ports have limited functionality: they can only receive data and notify owning clients of what data has been received.

<b>Compatibility</b>	Implemented only if <a href="#">New Serial Manager Feature Set Version 1</a> is present. If <a href="#">New Serial Manager Feature Set Version 2</a> is present, the SrmOpenBackground function is replaced by <a href="#">SrmExtOpenBackground</a> . SrmOpenBackground is supported for backward compatibility.
----------------------	---

<b>See Also</b>	<a href="#">SrmOpen</a>
-----------------	-------------------------

## SrmPrimeWakeupHandler

<b>Purpose</b>	Sets the number of received bytes that triggers a call to the wakeup handler function.					
<b>Declared In</b>	SerialMgr.h					
<b>Prototype</b>	Err SrmPrimeWakeupHandler (UInt16 portId, UInt16 minBytes)					
<b>Parameters</b>	<table><tr><td>-&gt; portId</td><td>Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a>.</td></tr><tr><td>-&gt; minBytes</td><td>Number of bytes that must be received before wakeup handler is called. Typically, this is set to 1.</td></tr></table>		-> portId	Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a> .	-> minBytes	Number of bytes that must be received before wakeup handler is called. Typically, this is set to 1.
-> portId	Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a> .					
-> minBytes	Number of bytes that must be received before wakeup handler is called. Typically, this is set to 1.					
<b>Result</b>	This function returns the following error codes: <table><tr><td>errNone</td><td>No error.</td></tr></table>		errNone	No error.		
errNone	No error.					

serErrBadPort This port doesn't exist.  
serErrNotOpen The port is not open.  
serErrNoDevicesAvail  
                  No serial devices could be found.

**Comments** This function primes a wakeup handler installed by [SrmSetWakeupHandler](#).

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**See Also** [SrmSetWakeupHandler](#), [WakeupHandlerProcPtr](#)

## SrmReceive

**Purpose** Receives a specified number of bytes.

**Declared In** SerialMgr.h

**Prototype** UInt32 SrmReceive (UInt16 portId, void \*rcvBufP,  
                      UInt32 count, Int32 timeout, Err \*errP)

**Parameters**

-> portID	Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a> .
<- rcvBufP	Pointer to buffer where received data is to be returned.
-> count	Length of data buffer (in bytes). This specifies the number of bytes to receive.
-> timeout	The amount of time (in ticks) that the Serial Manager waits to receive the requested block of data. At the end of the timeout, data received up to that time is returned.
<- errP	Error code.

**Result** Number of bytes of data actually received.

## Serial Manager

### *Serial Manager Functions*

---

- Comments** The following error codes can be returned in errP:
- |                           |  |
|---------------------------|--|
| errNone                   | No error.  |
| serErrBadPort             | This port doesn't exist.   |
| serErrNotOpen             | The port is not open.  |
| serErrTimeOut             | Unable to receive data within the specified timeout period.  |
| serErrConfigurationFailed | The port needs time to configure, and the configuration has failed.  |
| serErrNotSupported        | The port is not the foreground port.   |
| serErrConfigurationFailed | The port could not configure itself.   |
| serErrLineErr             | A line error occurred during the receipt of data. Use <a href="#">SrmGetStatus</a> to obtain the exact line error. |
| serErrNoDevicesAvail      | No serial devices could be found.  |
- Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.
- See Also** [SrmReceiveCheck](#), [SrmReceiveFlush](#), [SrmReceiveWait](#)

## SrmReceiveCheck

<b>Purpose</b>	Checks the receive FIFO and returns the number of bytes in the serial receive queue.									
<b>Declared In</b>	SerialMgr.h									
<b>Prototype</b>	Err SrmReceiveCheck (UInt16 portID, UInt32 *numBytesP)									
<b>Parameters</b>	<table><tr><td>-&gt; portID</td><td>Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a>.</td></tr><tr><td>&lt;- numBytesP</td><td>Number of bytes in the receive queue.</td></tr></table>		-> portID	Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a> .	<- numBytesP	Number of bytes in the receive queue.				
-> portID	Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a> .									
<- numBytesP	Number of bytes in the receive queue.									
<b>Result</b>	This function returns the following error codes: <table><tr><td>errNone</td><td>No error.</td></tr><tr><td>serErrBadPort</td><td>This port doesn't exist.</td></tr><tr><td>serErrNotOpen</td><td>The port is not open.</td></tr><tr><td>serErrLineErr</td><td>A line error has occurred. Use <a href="#">SrmGetStatus</a> to obtain the exact line error.</td></tr></table>		errNone	No error.	serErrBadPort	This port doesn't exist.	serErrNotOpen	The port is not open.	serErrLineErr	A line error has occurred. Use <a href="#">SrmGetStatus</a> to obtain the exact line error.
errNone	No error.									
serErrBadPort	This port doesn't exist.									
serErrNotOpen	The port is not open.									
serErrLineErr	A line error has occurred. Use <a href="#">SrmGetStatus</a> to obtain the exact line error.									
<b>Compatibility</b>	Implemented only if <a href="#">New Serial Manager Feature Set Version 1</a> is present.									
<b>See Also</b>	<a href="#">SrmReceive</a> , <a href="#">SrmReceiveFlush</a> , <a href="#">SrmReceiveWait</a>									

## SrmReceiveFlush

<b>Purpose</b>	Flushes the receive FIFOs.		
<b>Declared In</b>	SerialMgr.h		
<b>Prototype</b>	Err SrmReceiveFlush (UInt16 portId, Int32 timeout)		
<b>Parameters</b>	<table><tr><td>-&gt; portId</td><td>Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a>.</td></tr></table>	-> portId	Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a> .
-> portId	Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a> .		

## Serial Manager

### *Serial Manager Functions*

---

-> timeout      Timeout value, in ticks.

**Result**    This function returns the following error codes:

errNone      No error.

serErrBadPort      This port doesn't exist.

serErrNotOpen      The port is not open.

serErrNotSupported  
                    The port is not the foreground port.

serErrNoDevicesAvail  
                    No serial devices could be found.

**Comments**

The timeout value forces this function to wait a period of ticks after flushing the port to see if more data shows up to be flushed. If more data arrives within the timeout period, the port is flushed again and the timeout counter is reset and waits again. The function only exits after no more bytes are received by the port for the full timeout period since the last flush of the port. To avoid this waiting behavior, specify 0 for the timeout period.

Any errors on the line are cleared before this function returns.

**Compatibility**

Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**See Also**

[SrmReceive](#), [SrmReceiveCheck](#), [SrmReceiveWait](#)

## SrmReceiveWait

**Purpose** Waits until some number of bytes of data have arrived into the serial receive queue, then returns.

**Declared In** SerialMgr.h

**Prototype** Err SrmReceiveWait (UInt16 portId, UInt32 bytes, Int32 timeout)

**Parameters**

-> portID	Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a> .
-> bytes	Number of bytes to wait for.
-> timeout	Timeout value, in ticks.

**Result** This function returns the following error codes:

errNone	No error.
serErrBadPort	This port doesn't exist.
serErrNotOpen	The port is not open.
serErrTimeOut	Unable to receive data within the specified timeout period.
serErrNotSupported	The port is not the foreground port.
serErrBadParam	The bytes parameter exceeds the size of the receive queue. Use <a href="#">SrmSetReceiveBuffer</a> to increase the size of the receive queue.
serErrLineErr	A line error occurred during the receipt of data. Use <a href="#">SrmGetStatus</a> to obtain the exact line error.
serErrNoDevicesAvail	No serial devices could be found.

**Comments** If this function returns no error, the application can either check the number of bytes currently in the receive queue (using

## Serial Manager

### Serial Manager Functions

---

[SrmReceiveCheck](#)) or it can just specify a buffer and receive the data by calling [SrmReceive](#).

Do not call SerReceiveWait from within a wakeup handler. If you do, the serErrTimeOut error is returned.

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**See Also** [SrmReceive](#), [SrmReceiveCheck](#), [SrmReceiveFlush](#)

## SrmReceiveWindowClose

**Purpose** Closes direct access to the Serial Manager's receive queue.

**Declared In** SerialMgr.h

**Prototype** Err SrmReceiveWindowClose (UInt16 portId,  
                  UInt32 bytesPulled)

**Parameters** -> portId         Port ID returned from [SrmOpen](#) or  
                         [SrmExtOpen](#).  
-> bytesPulled     Number of bytes the application read from the  
                         receive queue.

**Result** This function returns the following error codes:

errNone	No error.
serErrBadPort	This port doesn't exist.
serErrNotOpen	The port is not open.
serErrNotSupported	The port is not the foreground port.
serErrNoDevicesAvail	No serial devices could be found.

**Comments** Call this function when the application has read as many bytes as it needs out of the receive queue or it has read all the available bytes.

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**See Also** [SrmReceiveWindowOpen](#)

## SrmReceiveWindowOpen

**Purpose** Provides direct access to the Serial Manager's receive queue.

**Declared In** SerialMgr.h

**Prototype** Err SrmReceiveWindowOpen (UInt16 portId,  
UInt8 \*\*bufPP, UInt32 \*sizeP)

**Parameters**

-> portId	Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a> .
<- bufPP	Pointer to a pointer to the receive buffer.
<- sizeP	Available bytes in buffer.

**Result** This function returns the following error codes:

errNone	No error.
serErrBadPort	This port doesn't exist.
serErrNotOpen	The port is not open.
serErrNotSupported	The port is not the foreground port.
serErrLineErr	The data in the queue contains line errors.
serErrNoDevicesAvail	No serial devices could be found.

**Comments** This function lets applications directly access the Serial Manager's receive queue to eliminate buffer copying by the Serial Manager. This access is a "back door" route to the received data. After retrieving data from the buffer, the application must call [SrmReceiveWindowClose](#).

## Serial Manager

### Serial Manager Functions

---

Applications that want to empty the receive buffer entirely should call the `SrmReceiveWindowOpen` and `SrmReceiveWindowClose` functions repeatedly until the buffer size returned is 0.

---

**IMPORTANT:** Once an application calls `SrmReceiveWindowOpen`, it should not attempt to receive data via the normal method of calling [`SrmReceive`](#) or [`SrmReceiveWait`](#), as these functions interfere with direct access to the receive queue.

---

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**See Also** [SrmReceiveWindowClose](#)

## SrmSend

**Purpose** Sends a block of data out the specified port.

**Declared In** `SerialMgr.h`

**Prototype** `UIInt32 SrmSend (UIInt16 portId, const void *bufP,  
                  UIInt32 count, Err *errP)`

**Parameters**

-> portID	Port ID returned from <a href="#"><u>SrmOpen</u></a> or <a href="#"><u>SrmExtOpen</u></a> .
-> bufP	Pointer to data to send.
-> count	Length of data buffer, in bytes.
<- errP	Error code. See the Comments section for details.

**Result** Number of bytes of data actually sent.

**Comments** When `SrmSend` returns, you should check the value returned in the `errP` parameter. If `errNone`, then the entire data buffer was sent. If

not `errNone`, then the result equals the number of bytes sent before the error occurred. The possible error values are:

<code>errNone</code>	No error.
<code>serErrBadPort</code>	This port doesn't exist.
<code>serErrNotOpen</code>	The port is not open.
<code>serErrTimeOut</code>	Unable to send data within the specified CTS timeout period.
<code>serErrNoDevicesAvail</code>	No serial devices could be found.
<code>serErrConfigurationFailed</code>	The port configuration has failed.
<code>serErrNotSupported</code>	The specified port is not the foreground port.

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**See Also** [SrmSendCheck](#), [SrmSendFlush](#), [SrmSendWait](#)

## SrmSendCheck

**Purpose** Checks the transmit FIFO and returns the number of bytes left to be sent.

**Declared In** `SerialMgr.h`

**Prototype** `Err SrmSendCheck (UInt16 portId,  
                  UInt32 *numBytesP)`

**Parameters** `-> portID` Port ID returned from [SrmOpen](#) or [SrmExtOpen](#).  
`<- numBytesP` Number of bytes left in the FIFO queue.

**Result** This function returns the following error codes:  
`errNone` No error.

## Serial Manager

### *Serial Manager Functions*

---

serErrBadPort This port doesn't exist.  
serErrNotOpen The port is not open.  
serErrNotSupported This feature not supported by the hardware.  
serErrNoDevicesAvail No serial devices could be found.

- Comments** Not all serial devices support this feature.
- Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.
- See Also** [SrmSend](#), [SrmSendFlush](#), [SrmSendWait](#)

## SrmSendFlush

- Purpose** Flushes the transmit FIFO.
- Declared In** SerialMgr.h
- Prototype** Err SrmSendFlush (UInt16 portId)
- Parameters** -> portID Port ID returned from [SrmOpen](#) or [SrmExtOpen](#).
- Result** This function returns the following error codes:
- errNone No error.  
serErrBadPort This port doesn't exist.  
serErrNotOpen The port is not open.  
serErrNotSupported The port is not the foreground port.  
serErrNoDevicesAvail No serial devices could be found.

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**See Also** [SrmSend](#), [SrmSendCheck](#), [SrmSendWait](#)

## SrmSendWait

**Purpose** Waits until all previous data has been sent from the transmit FIFO, then returns.

**Declared In** SerialMgr.h

**Prototype** Err SrmSendWait (UInt16 portId)

**Parameters** -> portID Port ID returned from [SrmOpen](#) or [SrmExtOpen](#).

**Result** This function returns the following error codes:

errNone No error.

serErrBadPort This port doesn't exist.

serErrNotOpen The port is not open.

serErrTimeOut Unable to send data within the CTS timeout period.

serErrNotSupported  
The port is not the foreground port.

serErrNoDevicesAvail  
No serial devices could be found.

**Comments** Consider calling this function if your software needs to detect when all data has been transmitted by [SrmSend](#). The SrmSend function blocks until all data has been transmitted or a timeout occurs. A subsequent call to SrmSendWait blocks until all data queued up for transmission has been transmitted or until another CTS timeout occurs (if CTS handshaking is enabled).

## Serial Manager

### Serial Manager Functions

---

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**See Also** [SrmSend](#), [SrmSendCheck](#), [SrmSendFlush](#)

## SrmSetReceiveBuffer

**Purpose** Installs a new buffer into the Serial Manager's receive queue.

**Declared In** SerialMgr.h

**Prototype** Err SrmSetReceiveBuffer (UInt16 portId,  
void \*bufP, UInt16 bufSize)

**Parameters**

-> portID	Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a> .
-> bufP	Pointer to new receive buffer. Ignored if bufSize is NULL.
-> bufSize	Size of new receive buffer in bytes. To remove this buffer and allocate a new default buffer (512 bytes), specify NULL.

**Result** This function returns the following error codes:

errNone	No error.
serErrBadPort	This port doesn't exist.
serErrNotOpen	This port is not open.
memErrNotEnoughSpace	Not enough memory to allocate default buffer.
serErrNoDevicesAvail	No serial devices could be found.

**Comments** The buffer that you pass to this function must remain allocated while you have the serial port open. Before you close the serial port, you must restore the default queue by calling SrmSetReceiveBuffer with NULL as the bufP and bufSize arguments.

**IMPORTANT:** Applications must install the default buffer before closing the port (or disposing of the new receive queue).

---

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

## SrmSetWakeupHandler

**Purpose** Installs a wakeup handler.

**Declared In** SerialMgr.h

**Prototype** Err SrmSetWakeupHandler (UInt16 portId,  
WakeupHandlerProcPtr procP, UInt32 refCon)

**Parameters**

-> portID	Port ID returned from <a href="#">SrmOpen</a> or <a href="#">SrmExtOpen</a> .
-> procP	Pointer to a <a href="#">WakeupHandlerProcPtr</a> function. Specify NULL to remove a handler.
-> refCon	User-defined data that is passed to the wakeup handler function. This can be a pointer or not.

**Result** This function returns the following error codes:

errNone	No error.
serErrBadPort	This port doesn't exist.
serErrNotOpen	The port is not open.
serErrNoDevicesAvail	No serial devices could be found.

**Comments** The wakeup handler is a function in your application that you want to be called whenever there is data ready to be received on the specified port.

The wakeup handler function will not become active until it is primed with a number of bytes that is greater than 0, by the

## Serial Manager

### Serial Manager Application-Defined Functions

---

[SrmPrimeWakeupHandler](#) function. Every time a wakeup handler is called, it must be re-primed (using [SrmPrimeWakeupHandler](#)) in order to be called again.

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**See Also** [SrmPrimeWakeupHandler](#), [WakeupHandlerProcPtr](#)

## Serial Manager Application-Defined Functions

### WakeupHandlerProcPtr

**Purpose** Called after some number of bytes are received by the Serial Manager's interrupt function.

**Declared In** SerialMgr.h

**Prototype** void (\*WakeupHandlerProcPtr) (UInt32 refCon)

**Parameters** ->refCon User-defined data passed from the [SrmSetWakeupHandler](#) function.

**Result** Returns nothing.

**Comments** This handler function is installed by calling [SrmSetWakeupHandler](#). The number of bytes after which it is called is specified by [SrmPrimeWakeupHandler](#).

---

**IMPORTANT:** Because wakeup handlers are called during interrupt time, they cannot call **any** Palm OS® system functions, including [SrmReceive](#), that may block the system in any way. Wakeup handlers should also be very short so as to reduce interrupt latency.

---

Two common implementations of wakeup handlers include:

- Calling [EvtWakeup](#), which causes any pending EvtGetEvent call to return and then sends a nilEvent to the current application.
- Using [SrmReceiveWindowOpen](#) and [SrmReceiveWindowClose](#) to gain direct access to the receive queue without blocking.

**Compatibility** Implemented only if [New Serial Manager Feature Set Version 1](#) is present.

**See Also** [SrmPrimeWakeupHandler](#), [SrmSetWakeupHandler](#)

## **Serial Manager**

*Serial Manager Application-Defined Functions*

---

# Old Serial Manager

---

This chapter provides reference material for the serial manager API:

- [Serial Manager Data Structures](#)
- [Serial Manager Functions](#)

The header file `SerialMgrOld.h` declares the serial manager API. For more information on the serial manager, see the chapter “[Serial Communication](#)” in the *Palm OS Programmer’s Companion*, vol. II, *Communications*.

---

**NOTE:** The API described in this chapter is obsolete if the [New Serial Manager Feature Set](#) is present. The API is still supported for backward compatibility; however, the Serial Manager APIs are preferred.

---

## Serial Manager Data Structures

### SerCtlEnum

To perform a control function, applications call [SerControl](#), which performs one of the control operations specified by `SerCtlEnum`, which has the following elements:

Element	Description
<code>serCtlFirstReserved = 0</code>	Reserve 0
<code>serCtlStartBreak</code>	Turn RS232 break signal on. Applications have to make sure that the break is set long enough to generate a value BREAK! <code>valueP = 0; valueLenP = 0</code>
<code>serCtlStopBreak</code>	Turn RS232 break signal off: <code>valueP = 0; valueLenP = 0</code>

## Old Serial Manager

### Serial Manager Data Structures

---

Element	Description
serCtlBreakStatus	Get RS232 break signal status (on or off): valueP = ptr to Word for returning status (0 = off, !0 = on)  *valueLenP = sizeof(Word)
serCtlStartLocalLoopback	Start local loopback test; valueP = 0, valueLenP = 0
serCtlStopLocalLoopback	Stop local loopback test valueP = 0, valueLenP = 0
serCtlMaxBaud	valueP = ptr to DWord for returned baud *valueLenP = sizeof(DWord)
serCtlHandshakeThreshold	Retrieve HW handshake threshold; this is the maximum baud rate that does not require hardware handshaking valueP = ptr to DWord for returned baud *valueLenP = sizeof(DWord)
serCtlEmuSetBlockingHook	Set a blocking hook routine.
<hr/> <b>WARNING!</b> WARNING: For use with the Simulator on Mac OS only: NOT SUPPORTED ON THE PALM DEVICE. <hr/>	
	valueP = ptr to SerCallbackEntryType *valueLenP = sizeof(SerCallbackEntryType) Returns the old settings in the first argument.
serCtlLAST	Add new address entries before this one.

## SerSettingsType

The `SerSettingsType` structure defines serial port attributes; it is used by the calls `SerGetSettings` and `SerSetSettings`. The `SerSettingsPtr` type points to a `SerSettingsType` structure.

```
typedef struct SerSettingsType {  
    UInt32 baudRate;
```

```
    UInt32 flags;
    Int32 ctsTimeout;
} SerSettingsType;

typedef SerSettingsType* SerSettingsPtr;
```

### Field Descriptions

baudRate	Baud rate
flags	Miscellaneous settings
ctsTimeout	Maximum number of ticks to wait for CTS to become asserted before transmitting; used only when configured with the serSettingsFlagCTSAutoM flag.

## Serial Manager Functions

### SerClearErr

**Purpose** Reset the serial port's line error status.

**Declared In** SerialMgrOld.h

**Prototype** Err SerClearErr (UInt16 refNum)

**Parameters** -> refNum The serial library reference number.

**Result** 0 No error.

**Comments** Call SerClearErr only after a serial manager function (SerReceive, SerReceiveCheck, SerSend, etc.) returns with the error code serErrLineErr.

The reason for this is that SerClearErr resets the serial port. So, if SerClearErr is called unconditionally while a byte is coming into the serial port, that byte is guaranteed to become corrupted.

## Old Serial Manager

### *Serial Manager Functions*

---

The right strategy is to always check the error code returned by a serial manager function. If it's `serErrLineErr`, call `SerClearErr` immediately. However, don't make unsolicited calls to `SerClearErr`.

When you get `serErrLineErr`, consider flushing the receive queue for a fraction of a second by calling `SerReceiveFlush`. `SerReceiveFlush` calls `SerClearErr` for you.

## **SerClose**

**Purpose** Release the serial port previously acquired by `SerOpen`.

**Declared In** `SerialMgrOld.h`

**Prototype** `Err SerClose (UInt16 refNum)`

**Parameters** `-> refNum`      Serial library reference number.

**Result** `0`      No error.

`serErrNotOpen`      Port wasn't open.

`serErrStillOpen`      Port still held open by another process.

**Comments** Releases the serial port and shuts down serial port hardware if the open count has reached 0. Open serial ports consume more energy from the device's batteries; it's therefore essential to keep a port open only as long as necessary.

**Caveat** Don't call `SerClose` unless the return value from `SerOpen` was 0 (zero) or `serErrAlreadyOpen`.

**See Also** [SerOpen](#)

## SerControl

<b>Purpose</b>	Perform a control function.	
<b>Declared In</b>	SerialMgrOld.h	
<b>Prototype</b>	<pre>Err SerControl (UInt16 refNum, UInt16 op, void *valueP, UInt16 *valueLenP)</pre>	
<b>Parameters</b>	-> refNum	Reference number of library.
	-> op	Control operation to perform (SerCtlEnum).
	<-> valueP	Pointer to value for operation.
	<-> valueLenP	Pointer to size of value.
<b>Result</b>	0	No error.
	serErrBadParam	Invalid parameter (unknown).
	serErrNotOpen	Library not open.
<b>Comments</b>	This function provides extensible control features for the serial manager. You can <ul style="list-style-type: none"><li>• Turn on/off the RS232 break signal and check its status.</li><li>• Perform a local loopback test.</li><li>• Get the maximum supported baud rate.</li><li>• Get the hardware handshake threshold baud rate.</li></ul>	
<b>Compatibility</b>	Implemented only if <a href="#">2.0 New Feature Set</a> is present.	

## Old Serial Manager

### Serial Manager Functions

---

## SerGetSettings

**Purpose** Fill in the [SerSettingsType](#) structure with current serial port attributes.

**Declared In** SerialMgrOld.h

**Prototype** Err SerGetSettings (UInt16 refNum,  
SerSettingsPtr settingsP)

**Parameters** -> refNum      Serial library reference number.  
    <-> settingsP      Pointer to [SerSettingsType](#) structure to be filled in.

**Result** 0      No error.  
              serErrNotOpen      The port wasn't open.

**Comments** The information returned by this call includes the current baud rate, CTS timeout, handshaking options, and data format options.  
See the [SerSettingsType](#) structure for more details.

**See Also** [SerSend](#)

## SerGetStatus

**Purpose** Return the pending line error status for errors that have been detected since the last time [SerClearErr](#) was called.

**Declared In** SerialMgrOld.h

**Prototype** UInt16 SerGetStatus (UInt16 refNum,  
Boolean \*ctsOnP, Boolean \*dsrOnP)

**Parameters** -> refNum      Serial library reference number.  
    -> ctsOnP      Pointer to location for storing a Boolean value.

-> dsrOnP              Pointer to location for storing a Boolean value.

**Result**    Returns any combination of the following constants, bitwise OR'ed together:

serLineErrorParity  
                    Parity error.

serLineErrorHWOVERRUN  
                    Hardware overrun.

serLineErrorFraming  
                    Framing error.

serLineErrorBreak  
                    Break signal detected.

serLineErrorHShake  
                    Line handshake error.

serLineErrorSWOVERRUN  
                    Software overrun.

**Comments**    When another serial manager function returns an error code of `serErrLineErr`, `SerGetStatus` can be used to find out the specific nature of the line error(s).

The values returned via `ctsOnP` and `dsrOnP` are not meaningful in the present version of the software

**See Also**    [SerClearErr](#)

## SerOpen

**Purpose**    Acquire and open a serial port with given baud rate and default settings.

**Declared In**    `SerialMgrOld.h`

**Prototype**    `Err SerOpen (UInt16 refNum, UInt16 port, UInt32 baud)`

**Parameters**    -> `refNum`              Serial library reference number.

## Old Serial Manager

### *Serial Manager Functions*

---

	-> port	Port number.
	-> baud	Baud rate.
<b>Result</b>	0	No error.
	serErrAlreadyOpen	Port was open. Enables port sharing by “friendly” clients (not recommended).
	serErrBadParam	Invalid parameter.
	memErrNotEnoughSpace	Insufficient memory.

#### **Comments**

Acquires the serial port, powers it up, and prepares it for operation. To obtain the serial library reference number, call [SysLibFind](#) with “Serial Library” as the library name. This reference number must be passed as a parameter to all serial manager functions. The device currently contains only one serial port with port number 0 (zero).

The baud rate is an integral baud value (for example - 300, 1200, 2400, 9600, 19200, 38400, 57600, etc.). The Palm OS® device has been tested at the standard baud rates in the range of 300 - 57600 baud. Baud rates through 1 Mbit are theoretically possible. Use CTS handshaking at baud rates above 19200 (see [SerSetSettings](#)).

An error code of 0 (zero) or `serErrAlreadyOpen` indicates that the port was successfully opened. If the port is already open when `SerOpen` is called, the port’s open count is incremented and an error code of `serErrAlreadyOpen` is returned. This ability to open the serial port multiple times allows cooperating tasks to share the serial port. Other tasks must refrain from using the port if `serErrAlreadyOpen` is returned and close it by calling [SerClose](#).

## SerReceive

**Purpose** Receives size bytes worth of data or returns with error if a line error or timeout is encountered.

**Declared In** SerialMgrOld.h

**Prototype** `UInt32 SerReceive (UInt16 refNum, void *bufP,  
                  UInt32 count, Int32 timeout, Err* errP)`

**Parameters**

<code>refNum</code>	Serial library reference number.
<code>&lt;-&gt; bufP</code>	Buffer for receiving data.
<code>-&gt; count</code>	Number of bytes to receive.
<code>-&gt; timeout</code>	Interbyte timeout in ticks, 0 for none, -1 forever.
<code>&lt;-&gt; errP</code>	For returning error code.

**Result** Number of bytes received:

<code>*errP = 0</code>	No error.
<code>serErrLineErr</code>	RS232 line error.
<code>serErrTimeOut</code>	Interbyte timeout.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

---

**NOTE:** The old versions of `SerSend` and `SerReceive` are still available as `SerSend10` and `SerReceive10` (not V10).

---

**See Also** [SerReceive10](#)

## Old Serial Manager

### *Serial Manager Functions*

---

## SerReceive10

<b>Purpose</b>	Receive a stream of bytes.	
<b>Declared In</b>	SerialMgrOld.h	
<b>Prototype</b>	<pre>Err SerReceive10 (UInt16 refNum, void *bufP,                   UInt32 bytes, Int32 timeout)</pre>	
<b>Parameters</b>	-> refNum	The serial library reference number.
	-> bufP	Pointer to the buffer for receiving data.
	-> bytes	Number of bytes desired.
	-> timeout	Interbyte time out in system ticks (-1 = forever).
<b>Result</b>	0	No error. Requested number of bytes was received.
	serErrTimeOut	Interbyte time out exceeded while waiting for the next byte to arrive.
	serErrLineErr	Line error occurred (see <a href="#">SerClearErr</a> and <a href="#">SerGetStatus</a> ).
<b>Comments</b>	SerReceive blocks until all the requested data has been received or an error occurs. Because this call returns immediately without any data if line errors are pending, it is important to acknowledge the detection of line errors by calling <a href="#">SerClearErr</a> . If you just need to retrieve all or some of the bytes which are already in the receive queue, call <a href="#">SerReceiveCheck</a> first to get the count of bytes presently in the receive queue.	
<b>Compatibility</b>	This function corresponds to the 1.0 version of SerReceive.	

## SerReceiveCheck

<b>Purpose</b>	Return the count of bytes presently in the receive queue.	
<b>Declared In</b>	SerialMgrOld.h	
<b>Prototype</b>	Err SerReceiveCheck (UInt16 refNum, UInt32 *numBytesP)	
<b>Parameters</b>	-> refNum	Serial library reference number.
	<-> numBytesP	Pointer to location for returning the byte count.
<b>Result</b>	0	No error.
	serErrLineErr	Line error pending (see <a href="#">SerClearErr</a> and <a href="#">SerGetStatus</a> ).
<b>Comments</b>	Because this call does not return the byte count if line errors are pending, it is important to acknowledge the detection of line errors by calling <a href="#">SerClearErr</a> .	
<b>See Also</b>	<a href="#">SerReceiveWait</a>	

## SerReceiveFlush

<b>Purpose</b>	Discard all data presently in the receive queue and flush bytes coming into the serial port. Clear the saved error status.	
<b>Declared In</b>	SerialMgrOld.h	
<b>Prototype</b>	void SerReceiveFlush (UInt16 refNum, Int32 timeout)	
<b>Parameters</b>	-> refNum	Serial library reference number.
	-> timeout	Interbyte time out in system ticks (-1 = forever).
<b>Result</b>	Returns nothing.	

## Old Serial Manager

### *Serial Manager Functions*

---

**Comments** SerReceiveFlush blocks until a timeout occurs while waiting for the next byte to arrive.

## SerReceiveWait

**Purpose** Wait for at least bytes bytes of data to accumulate in the receive queue.

**Declared In** SerialMgrOld.h

**Prototype** Err SerReceiveWait (UInt16 refNum, UInt32 bytes, Int32 timeout)

**Parameters**

-> refNum	Serial library reference number.
-> bytes	Number of bytes desired.
-> timeout	Interbyte timeout in system ticks (-1 = forever).

**Result**

0	No error.
serErrTimeOut	Interbyte timeout exceeded while waiting for next byte to arrive.
serErrLineErr	Line error occurred (see <a href="#">SerClearErr</a> and <a href="#">SerGetStatus</a> ).

**Comments** This is the preferred method of waiting for serial input, since it blocks the current task and allows switching the processor into a more energy-efficient state.

SerReceiveWait blocks until the desired number of bytes accumulate in the receive queue or an error occurs. The desired number of bytes must be less than the current receive queue size. The default queue size is 512 bytes. Because this call returns immediately if line errors are pending, it is important to acknowledge the detection of line errors by calling [SerClearErr](#).

**See Also** [SerReceiveCheck](#), [SerSetReceiveBuffer](#)

## SerSend

**Purpose** Send one or more bytes of data over the serial port.

**Declared In** SerialMgrOld.h

**Prototype** `UInt32 SerSend (UInt16 refNum, void *bufP,  
                  UInt32 count, Err *errP)`

**Parameters**

-> refNum	Serial library reference number.
-> bufP	Pointer to data to send.
-> count	Number of bytes to send.
<-> errP	For returning error code.

**Result** Returns the number of bytes transferred.

Stores in `errP`:

0 No error.

`serErrTimeOut` Handshake timeout.

The old calls worked, but they did not return enough info when they failed. The new calls (available in Palm OS v2.0 and greater) add more parameters to solve this problem and make serial communications programming simpler.

Don't call the new functions when running on Palm OS 1.0.

**Compatibility** Implemented only if [2.0 New Feature Set](#) is present.

---

**NOTE:** The old versions of `SerSend` and `SerReceive` are still available as `SerSend10` and `SerReceive10` (not V10).

---

**See Also** [SerSend10](#), [SerSendWait](#)

## SerSend10

**Purpose** Send a stream of bytes to the serial port.

**Declared In** SerialMgrOld.h

**Prototype** Err SerSend10 (UInt16 refNum, void \*bufP,  
                  UInt32 size)

**Parameters**

-> refNum	Serial library reference number.
-> bufP	Pointer to the data to send.
-> size	Size (in number of bytes) of the data to send.

**Result**

0	No error.
serErrTimeOut	Handshake timeout (such as waiting for CTS to become asserted).

**Comments** In the present implementation, `SerSend10` blocks until all data is transferred to the UART or a timeout error (if CTS handshaking is enabled) occurs. Future implementations may queue up the request and return immediately, performing transmission in the background. If your software needs to detect when all data has been transmitted, see [SerSendWait](#).  
This routine observes the current CTS time out setting if CTS handshaking is enabled (see [SerGetSettings](#) and [SerSend](#)).

**Compatibility** This function corresponds to the 1.0 version of `SerSend`.

**See Also** [SerSend](#), [SerSendWait](#)

## SerSendFlush

**Purpose** Discard all data presently in the transmit queue.

**Declared In** SerialMgrOld.h

**Prototype** Err SerSendFlush (UInt16 refNum)

**Parameters** -> refNum      Serial library reference number.

**Result** 0      No error.

**See Also** [SerSend](#), [SerSendWait](#)

## SerSendWait

**Purpose** Wait until the serial transmit buffer empties.

**Declared In** SerialMgrOld.h

**Prototype** Err SerSendWait (UInt16 refNum, Int32 timeout)

**Parameters** -> refNum      Serial library reference number.  
-> timeout      Reserved for future enhancements. Set to (-1) for compatibility.

**Result** 0      No error.

serErrTimeOut      Handshake timeout (such as waiting for CTS to become asserted).

**Comments** SerSendWait blocks until all data is transferred or a timeout error (if CTS handshaking is enabled) occurs. This routine observes the current CTS timeout setting if CTS handshaking is enabled (see [SerGetSettings](#) and [SerSend](#)).

**See Also** [SerSend](#)

## Old Serial Manager

### *Serial Manager Functions*

---

## SerSetReceiveBuffer

**Purpose** Replace the default receive queue. To restore the original buffer, pass `bufSize = 0`.

**Declared In** `SerialMgrOld.h`

**Prototype** `Err SerSetReceiveBuffer (UInt16 refNum,  
void *bufP, UInt16 bufSize)`

**Parameters**

<code>-&gt; refNum</code>	Serial library reference number.
<code>-&gt; bufP</code>	Pointer to buffer to be used as the new receive queue.
<code>-&gt; bufSize</code>	Size of buffer, or 0 to restore the default receive queue.

**Result** Returns 0 if successful.

**Comments** The specified buffer needs to contain 32 extra bytes for serial manager overhead (its size should be your application's requirement plus 32 bytes). The default receive queue must be restored before the serial port is closed. To restore the default receive queue, call [SerSetReceiveBuffer](#) passing 0 (zero) for the buffer size. The serial manager does not free the custom receive queue.

## SerSetSettings

**Purpose** Set the serial port settings; that is, change its attributes.

**Declared In** `SerialMgrOld.h`

**Prototype** `Err SerSetSettings (UInt16 refNum,  
SerSettingsPtr settingsP)`

**Parameters**

<code>-&gt; refNum</code>	Serial library reference number.
---------------------------	----------------------------------

<-> settingsP      Pointer to the filled in [SerSettingsType](#) structure.

**Result**    0                  No error.

serErrNotOpen    The port wasn't open.

serErrBadParam    Invalid parameter.

**Comments**    The attributes set by this call include the current baud rate, CTS timeout, handshaking options, and data format options. See the definition of the [SerSettingsType](#) structure for more details.

To do 7E1 transmission, OR together:

```
serSettingsFlagBitsPerChar7 |  
serSettingsFlagParityOnM |  
serSettingsFlagParityEvenM |  
serSettingsFlagStopBits1
```

If you're trying to communicate at speeds greater than 19.2 Kbps, you need to use hardware handshaking:

```
serSettingsFlagRTSAutoM | serSettingsFlagCTSAutoM.
```

**See Also**    [SerGetSettings](#)

## **Old Serial Manager**

### *Serial Manager Functions*

---

# Serial Link Manager

---

This chapter provides reference material for the serial link manager API. The header file `SerialLinkMgr.h` declares the serial link manager API. For more information on the serial link manager, see the chapter “[Serial Communication](#)” in the *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## Serial Link Manager Functions

### **SlkClose**

**Purpose** Close down the serial link manager.

**Declared In** `SerialLinkMgr.h`

**Prototype** `Err SlkClose (void)`

**Parameters** None.

**Result** 0 No error.

`slkErrNotOpen` The serial link manager was not open.

**Comments** When the open count reaches zero, this routine frees resources allocated by serial link manager.

## **SlkCloseSocket**

<b>Purpose</b>	Closes a socket previously opened with <a href="#">SlkOpenSocket</a> . The caller is responsible for closing the communications library used by this socket, if necessary.
<b>Declared In</b>	SerialLinkMgr.h
<b>Prototype</b>	Err SlkCloseSocket (UInt16 socket)
<b>Parameters</b>	socket                    The socket ID to close.
<b>Result</b>	0                        No error. slkErrSocketNotOpen The socket was not open.
<b>Comments</b>	SlkCloseSocket frees system resources the serial link manager allocated for the socket. It does not free resources allocated and passed by the client, such as the buffers passed to <a href="#">SlkSetSocketListener</a> ; this is the client's responsibility. The caller is also responsible for closing the communications library used by this socket.
<b>See Also</b>	<a href="#">SlkOpenSocket</a>

## **SlkFlushSocket**

<b>Purpose</b>	Flush the receive queue of the communications library associated with the given socket.
<b>Declared In</b>	SerialLinkMgr.h
<b>Prototype</b>	Err SlkFlushSocket (UInt16 socket, Int32 timeout)
<b>Parameters</b>	-> socket                    Socket ID.

-> timeout      Interbyte timeout in system ticks.

**Result** 0      No error.

slkErrSocketNotOpen  
The socket wasn't open.

## SlkOpen

**Purpose** Initialize the serial link manager.

**Declared In** SerialLinkMgr.h

**Prototype** Err SlkOpen (void)

**Parameters** None.

**Result** 0      No error.

slkErrAlreadyOpen  
No error.

**Comments** Initializes the serial link manager, allocating necessary resources. Return codes of 0 (zero) and slkErrAlreadyOpen both indicate success. Any other return code indicates failure. The slkErrAlreadyOpen function informs the client that someone else is also using the serial link manager. If the serial link manager was successfully opened by the client, the client needs to call [SlkClose](#) when it finishes using the serial link manager.

## SlkOpenSocket

<b>Purpose</b>	Open a serial link socket and associate it with a communications library. The socket may be a known static socket or a dynamically assigned socket.							
<b>Declared In</b>	SerialLinkMgr.h							
<b>Prototype</b>	<pre>Err SlkOpenSocket (UInt16 portID,                   UInt16 *socketP, Boolean staticSocket)</pre>							
<b>Parameters</b>	<table><tr><td>portID</td><td>Comm library reference number for socket.</td></tr><tr><td>socketP</td><td>Pointer to location for returning the socket ID.</td></tr><tr><td>staticSocket</td><td>If TRUE, *socketP contains the desired static socket number to open. If FALSE, any free socket number is assigned dynamically and opened.</td></tr></table>		portID	Comm library reference number for socket.	socketP	Pointer to location for returning the socket ID.	staticSocket	If TRUE, *socketP contains the desired static socket number to open. If FALSE, any free socket number is assigned dynamically and opened.
portID	Comm library reference number for socket.							
socketP	Pointer to location for returning the socket ID.							
staticSocket	If TRUE, *socketP contains the desired static socket number to open. If FALSE, any free socket number is assigned dynamically and opened.							
<b>Result</b>	0	No error.						
	slkErrOutOfSockets	No more sockets can be opened.						
<b>Comments</b>	The communications library must already be initialized and opened (see <a href="#">SerOpen</a> ). When finished using the socket, the caller must call <a href="#">SlkCloseSocket</a> to free system resources allocated for the socket. For information about well-known static socket IDs, see <a href="#">The Serial Link Protocol</a> .							

## SlkReceivePacket

<b>Purpose</b>	Receive and validate a packet for a particular socket or for any socket. Check for format and checksum errors.												
<b>Declared In</b>	<code>SerialLinkMgr.h</code>												
<b>Prototype</b>	<code>Err SlkReceivePacket (UInt16 socket, Boolean andOtherSockets, SlkPktHeaderPtr headerP, void* bodyP, UInt16 bodySize, Int32 timeout)</code>												
<b>Parameters</b>	<table><tr><td>-&gt; socket</td><td>The socket ID.</td></tr><tr><td>-&gt; andOtherSockets</td><td>If TRUE, ignore destination in packet header.</td></tr><tr><td>&lt;-&gt; headerP</td><td>Pointer to the packet header buffer (size of <code>SlkPktHeaderType</code>).</td></tr><tr><td>&lt;-&gt; bodyP</td><td>Pointer to the packet client data buffer.</td></tr><tr><td>-&gt; bodySize</td><td>Size of the client data buffer (maximum client data size which can be accommodated).</td></tr><tr><td>-&gt; timeout</td><td>Maximum number of system ticks to wait for beginning of a packet; -1 means wait forever.</td></tr></table>	-> socket	The socket ID.	-> andOtherSockets	If TRUE, ignore destination in packet header.	<-> headerP	Pointer to the packet header buffer (size of <code>SlkPktHeaderType</code> ).	<-> bodyP	Pointer to the packet client data buffer.	-> bodySize	Size of the client data buffer (maximum client data size which can be accommodated).	-> timeout	Maximum number of system ticks to wait for beginning of a packet; -1 means wait forever.
-> socket	The socket ID.												
-> andOtherSockets	If TRUE, ignore destination in packet header.												
<-> headerP	Pointer to the packet header buffer (size of <code>SlkPktHeaderType</code> ).												
<-> bodyP	Pointer to the packet client data buffer.												
-> bodySize	Size of the client data buffer (maximum client data size which can be accommodated).												
-> timeout	Maximum number of system ticks to wait for beginning of a packet; -1 means wait forever.												
<b>Result</b>	<table><tr><td>0</td><td>No error.</td></tr><tr><td><code>slkErrSocketNotOpen</code></td><td>The socket was not open.</td></tr><tr><td><code>slkErrTimeOut</code></td><td>Timed out waiting for a packet.</td></tr><tr><td><code>slkErrWrongDestSocket</code></td><td>The packet being received had an unexpected destination.</td></tr><tr><td><code>slkErrChecksum</code></td><td>Invalid header checksum or packet CRC-16.</td></tr><tr><td><code>slkErrBuffer</code></td><td>Client data buffer was too small for packet's client data.</td></tr></table>	0	No error.	<code>slkErrSocketNotOpen</code>	The socket was not open.	<code>slkErrTimeOut</code>	Timed out waiting for a packet.	<code>slkErrWrongDestSocket</code>	The packet being received had an unexpected destination.	<code>slkErrChecksum</code>	Invalid header checksum or packet CRC-16.	<code>slkErrBuffer</code>	Client data buffer was too small for packet's client data.
0	No error.												
<code>slkErrSocketNotOpen</code>	The socket was not open.												
<code>slkErrTimeOut</code>	Timed out waiting for a packet.												
<code>slkErrWrongDestSocket</code>	The packet being received had an unexpected destination.												
<code>slkErrChecksum</code>	Invalid header checksum or packet CRC-16.												
<code>slkErrBuffer</code>	Client data buffer was too small for packet's client data.												
	If <code>andOtherSockets</code> is FALSE, this routine returns with an error code unless it gets a packet for the specific socket.												

## **Serial Link Manager**

### *Serial Link Manager Functions*

---

If `andOtherSockets` is TRUE, this routine returns successfully if it sees any incoming packet from the communications library used by `socket`.

<b>Comments</b>	You may request to receive a packet for the passed socket ID only, or for any open socket which does not have a socket listener. The parameters also specify buffers for the packet header and client data, and a timeout. The timeout indicates how long the receiver should wait for a packet to begin arriving before timing out. If a packet is received for a socket with a registered socket listener, it will be dispatched via its socket listener procedure. On success, the packet header buffer and packet client data buffer is filled in with the actual size of the packet's client data in the packet header's <code>bodySize</code> field.
-----------------	--

## **SlkSendPacket**

**Purpose** Send a serial link packet via the serial output driver.

**Declared In** `SerialLinkMgr.h`

**Prototype** `Err SlkSendPacket (SlkPktHeaderPtr headerP,  
SlkWriteDataPtr writeList)`

**Parameters** `<-> headerP` Pointer to the packet header structure with client information filled in (see Comments).  
`-> writeList` List of packet client data blocks (see Comments).

**Result** 0 No error.

`slkErrSocketNotOpen`  
The socket was not open.

`slkErrTimeOut` Handshake timeout.

**Comments** `SlkSendPacket` stuffs the signature, client data size, and the checksum fields of the packet header. The caller must fill in all other packet header fields. If the transaction ID field is set to 0 (zero), the

serial link manager automatically generates and stuffs a new non-zero transaction ID. The array of SlkWriteDataType structures enables the caller to specify the client data part of the packet as a list of noncontiguous blocks. The end of list is indicated by an array element with the size field set to 0 (zero). This call blocks until the entire packet is sent out or until an error occurs.

## **SlkSetSocketListener**

**Purpose** Register a socket listener for a particular socket.

**Declared In** SerialLinkMgr.h

**Prototype** Err SlkSetSocketListener (UInt16 socket,  
SlkSocketListenPtr socketP)

**Parameters** -> socket      Socket ID.  
-> socketP      Pointer to a SlkSocketListenType  
structure.

**Result** 0      No error.

slkErrBadParam      Invalid parameter.

slkErrSocketNotOpen  
The socket was not open.

**Comments** Called by applications to set up a socket listener.

Since the serial link manager does not make a copy of the SlkSocketListenType structure, but instead saves the passed pointer to it, the structure

- must **not** be an automatic variable (that is, local variable allocated on the stack)
- may be a global variable in an application
- may be a locked chunk allocated from the dynamic heap

The SlkSocketListenType structure specifies pointers to the socket listener procedure and the data buffers for dispatching packets destined for this socket. Pointers to two buffers must be

## Serial Link Manager

### Serial Link Manager Functions

---

specified: the packet header buffer (size of `SlkPktHeaderType`), and the packet body (client data) buffer. The packet body buffer must be large enough for the largest expected client data size. Both buffers may be application global variables or locked chunks allocated from the dynamic heap.

The socket listener procedure is called when a valid packet is received for the socket. Pointers to the packet header buffer and the packet body buffer are passed as parameters to the socket listener procedure.

---

**NOTE:** The application is responsible for freeing the `SlkSocketListenType` structure or the allocated buffers when the socket is closed. The serial link manager doesn't do it.

---

**Compatibility** If [5.0 New Feature Set](#) is present this function is unimplemented.

## SlkSocketPortID

**Purpose** Get the port ID associated with a particular socket; for use with the new serial manager.

**Declared In** `SerialLinkMgr.h`

**Prototype** `ErrSlkSocketPortID (UInt16 socket,  
                  UInt16 * portIDP)`

**Parameters** `-> socket` The socket ID.

`<-> portIDP` Pointer to location for returning the port ID.

**Result** 0 No error.

`slkErrSocketNotOpen`  
The socket was not open.

**Compatibility** Implemented only if [New Serial Manager Feature Set](#) is present.

## **SlkSocketSetTimeout**

**Purpose** Set the interbyte packet receive-timeout for a particular socket.

**Declared In** SerialLinkMgr.h

**Prototype** Err SlkSocketSetTimeout (UInt16 socket,  
Int32 timeout)

**Parameters**

-> socket	Socket ID.
-> timeout	Interbyte packet receive-timeout in system ticks.

**Result** 0 No error.

slkErrSocketNotOpen  
The socket was not open.

## **Serial Link Manager**

### *Serial Link Manager Functions*

---

# Telephony Basic Services

---

This chapter provides reference material for the Telephony API, which you can use to interface with telephone systems and equipment. This chapter discusses the following topics:

- [Telephony Data Structures](#)
- [Telephony Constants](#)
- [Telephony Functions](#)
- [Feature Support Functions](#)

The header file `TelephonyMgr.h` declares the telephony manager API. The header file `TelephonyMgrType.h` declares the data structures that you use with the telephony manager API.

For more information about using the telephony manager, see [Chapter 10, “Telephony Manager”](#) in *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## Telephony Service Types

The telephony API organizes functions within sets called **service sets**. Each service set contains a related set of functions that may or may not be available on a particular mobile device or network. You can use the [`TelIs<ServiceSet>Available`](#) function to determine if a service set is supported in the current environment, and you can use the [`TelIs<FunctionName>Supported`](#) to determine if a specific function is supported in the current environment.

## Telephony Basic Services

### Telephony Service Types

---

The telephony API documentation has been split into several chapters. Each chapter covers one or more of the service sets, as shown in [Table 68.1](#)

**Table 68.1 Telephony service types**

Service prefix	Functionality	Chapter	Description
Tel	Basic	<a href="#">Chapter 68, "Telephony Basic Services."</a>	Basic functions that are always available.
TelCfg	Configuration	<a href="#">Chapter 69, "Telephony Security and Configuration."</a>	Allows configuration of the phone, including Short Message Services (SMS) configuration.
TelDtc	Data calls	<a href="#">Chapter 71, "Telephony Calls."</a>	Functions for handling data calls.
TelEmc	Emergency calls	<a href="#">Chapter 71, "Telephony Calls."</a>	Functions for handling emergency calls.
TelInf	Information	<a href="#">Chapter 68, "Telephony Basic Services."</a>	Functions for retrieving information about the phone.
TelNwk	Network	<a href="#">Chapter 70, "Telephony Network."</a>	Provides network oriented services, including authorization, signal level, search mode, and related operations.
TelOem	OEM	<a href="#">Chapter 68, "Telephony Basic Services."</a>	Provides OEMs with the ability to incorporate custom functionality.
TelPhb	Phone book	<a href="#">Chapter 73, "Telephony Phone Book."</a>	Functions for managing the phone book.

**Table 68.1 Telephony service types (*continued*)**

Service prefix	Functionality	Chapter	Description
TelPow	Power	<a href="#">Chapter 68, “Telephony Basic Services.”</a>	Provides access to power supply level.
TelSms	Short Message Service	<a href="#">Chapter 72, “Telephony SMS.”</a>	Addresses the SMS, including functions for reading, sending, replying to, and deleting short messages.
TelSnd	Sound	<a href="#">Chapter 68, “Telephony Basic Services.”</a>	Functions for playing key tones on and muting the phone.
TelSpc	Speech calls	<a href="#">Chapter 71, “Telephony Calls.”</a>	Function for handling voice calls, including dual tone modulated frequency (DTMF) sounds.
TelSty	Security	<a href="#">Chapter 69, “Telephony Security and Configuration.”</a>	Supports PIN code management for access to phone and Subscriber Identity Module (SIM) security-related features.

## Telephony Data Structures

This section describes the data structures used with the basic services portion of the telephony API.

### TelEventType

The [TelGetEvent](#) and [TelGetTelephonyEvent](#) functions both return a TelEventType structure to provide information about a telephony-related event.

You call the [TelGetEvent](#) function to retrieve telephony and other events.

## Telephony Basic Services

### Telephony Data Structures

---

You call the [TelGetTelephonyEvent](#) function to retrieve only telephony events. This function does not consume non-telephony events.

```
typedef struct _TelEventType
{
    eventsEnum    eType;
    Boolean       penDown;
    UInt8         tapCount;
    Int16         screenX;
    Int16         screenY;
    UInt16        functionId;
    UInt16        transId;
    MemPtr        *paramP;
    Err           returnCode;
} TelEventType;
```

#### Field Descriptions

eType	One of the <a href="#">eventsEnum</a> constants. Specifies the type of the event.
penDown	true if the pen was down at the time of the event, and false if the pen was up.  Note that this field is not filled in for telephony events.
tapCount	The number of taps received at this location. This value is used mainly by text fields. When the user taps in a text field, two taps selects a word, and three taps selects the entire line.  Note that this field is not filled in for telephony events.

screenX	Window-relative position of the pen in pixels (number of pixels from the left bound of the window).  Note that this field is not filled in for telephony events.
screenY	Window-relative position of the pen in pixels (number of pixels from the top left of the window).  Note that this field is not filled in for telephony events.
functionId	The ID of the message associated with the function call, which specifies the telephony manager function that generated this event.
transId	The transaction ID that was associated with this asynchronous function call.
paramP	A pointer to a parameter structure that was passed when an asynchronous call was made.
returnCode	The return code of the asynchronously called function. The value of this field is errNone if the function succeeded, or an error code if the function failed.

Note that the first five fields of the TelEvent Type structure are the same as the first five fields of the [EventType](#) structure, which is described in [Chapter 2, “Palm OS Events.”](#)

## TelCallStateType

The [TelGetCallState](#) function uses the TelCallState structure to retrieve information about the current state of the connected phone.

```
typedef struct _TelGetCallStateType
{
```

## Telephony Basic Services

### Telephony Data Structures

---

```
    UInt8      state;
    UInt8      callType;
    UInt8      callServiceType;
    UInt8      numberSize;
    Char       *number;
} TelGetCallStateType
```

#### Field Descriptions

- <- state Filled in with one of the [Telephone Call State Constants](#), which indicates the current state of the telephone call.
- <- callType Filled in with one of the [Telephone Call Type Constants](#), which indicates the call type of the current telephone call.
- <- callServiceType Filled in with one of the [Telephone Call Service Type Constants](#), which indicates the call service type of the current telephone call.

<-> numberSize

The size of the number string buffer. When the structure is used as an input parameter, this is the allocated size, in bytes, of the buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the number buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the data, and the function using this structure generates a `telErrBufferSize` error.

<- number

A buffer into which the telephone number string is stored.

When the structure describes an incoming telephone call, this is the incoming telephone number. When the structure describes an outgoing telephone call, this is the telephone number that has been called.

Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a `telErrBufferSize` error.

## TelInfGetformationType

The [TelInfGetInformation](#) function uses the `TelInfGetInformationType` structure to retrieve information about the phone with which you are communicating.

```
typedef struct _TelInfGetInformationType
{
    UInt8      infoType;
    UInt8      size;
    UInt8      *value;
} TelInfGetInformationType
```

## Telephony Basic Services

### Telephony Data Structures

---

#### Field Descriptions

- > infoType      The type of information to retrieve. This must be one of the [Information Type Constants](#).
- <-> size      The size of the value buffer. When the structure is used as an input parameter, this is the allocated size, in bytes, of the buffer.  
  
Upon return, this is the actual size of the buffer. If the value buffer is too small to contain all of the retrieved information, this field is assigned the entire length of the data, and the function using this structure generates a `telErrBufferSize` error.
- <- value      A buffer into which the information is stored.

## TelOemCallType

You use the `TelOemCallType` to specify a [TelOemCall](#) function.

```
typedef struct _TelOemCallType
{
    UInt32    OemID;
    UInt8     funcID;
    void      *paramP;
} TelOemCallType
```

#### Field Descriptions

- > OemID      The unique ID of the OEM function set.
- > funcID      The ID of the function within the OEM function set.
- <-> paramP      A pointer to a parameter block that is passed to the OEM function. The OEM function might modify some of the fields in the parameter block.

## TelSendCommandStringType

The [TelSendCommandString](#) function uses the `TelSendCommandStringType` structure to send a command string.

```
typedef struct _TelSendCommandStringType
{
    Char     *commandString;
    Char     *resultString;
    UInt16   resultSize;
    UInt32   timeOut;
} TelSendCommandStringType
```

### Field Descriptions

- |     |               |  |
|-----|---------------|--|
| ->  | commandString | The command string to send.  |
| <-  | resultString  | The result string.   |
| <-> | resultSize    | The size of the resultString string buffer. When the structure is used as an input parameter, this is the allocated size, in bytes, of the buffer.<br><br>Upon return, this is the actual size of the string, including the null terminator character. If the resultString buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the data, and the function using this structure generates a telErrBufferSize error. |
| ->  | timeOut       | The number of milliseconds before timing out.  |

### TelSndPlayKeyToneType

The [TelSndPlayKeyTone](#) function uses the TelSndPlayKeyToneType structure to specify a key tone.

```
typedef struct _TelSndPlayKeyToneType
{
    UInt8     keycode;
    UInt8     type;
} TelSndPlayKeyToneType
```

## Telephony Basic Services

### Telephony Constants

---

#### Field Descriptions

- > keycode The keycode of the key tone to play. This must be one of the [Keycode Constants](#).
- > type The tone type. This must be one of the [Key Sound Type Constants](#).

## Telephony Constants

This section describes the data structures used with the basic services portion of the telephony API, which include the following constant types:

- [Battery Status Constants](#)
- [Telephone Call State Constants](#)
- [Telephone Call Type Constants](#)
- [Telephone Call Service Type Constants](#)
- [Error Code Constants](#)
- [Information Type Constants](#)
- [Keycode Constants](#)
- [Key Sound Type Constants](#)
- [Message Identifier Constants](#)
- [Service Set Constants](#)

### Battery Status Constants

The battery status constants provide information about the phone's battery.

Constant	Value	Description
kTelPowBatteryPowered	0	The phone is powered by a battery.
kTelPowBatteryNotPowered	1	The phone has a battery connected to it, but is not using that battery.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
kTelPowNobattery	2	The phone does not have a battery connected to it.
kTelPowBatteryFault	3	The phone has a recognized power fault; calls are currently inhibited.

## Telephone Call State Constants

The [TelCallStateType](#) structure uses the telephone call state constants to encode the current state of the connected telephone call.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
kTelCallIdle	0x00	The connection is idle.
kTelCallConnecting	0x01	A telephone call is currently connecting.
kTelCallConnected	0x02	A telephone call is currently connected.
kTelCallRedial	0x03	A telephone call is being re-dialed.
kTelCallIncoming	0x04	A telephone call is currently incoming.
kTelCallIncomingAck	0x05	An incoming telephone call is currently being acknowledged.
kTelCallDisconnecting	0x06	A telephone call is being disconnected.

## Telephone Call Type Constants

The [TelCallStateType](#) structure uses the telephone call type constants to encode the type of the current telephone call.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
kTelCallTypeOutgoing	0x00	An outgoing telephone call.
kTelCallTypeIncoming	0x01	An incoming telephone call.

## Telephony Basic Services

### Telephony Constants

---

## Telephone Call Service Type Constants

The [TelCallStateType](#) structure uses the telephone call service type constants to encode the service type of the current telephone call.

Constant	Value	Description
kTelCallServiceVoice	0x00	A voice telephone call.
kTelCallServiceData	0x01	A data telephone call.

## Error Code Constants

The telephony manager functions return the error code constants shown in the following table to indicate their status.

Constant	Description
telErrBufferSize	One of the buffers used to retrieve data is too small.
telErrCodingScheme	The coding scheme specified for the short message is not valid.
telErrCommandFailed	The specified command could not be performed by the phone. Check the phone driver.
telErrEntryNotFound	The specified entry was not found.
telErrFeatureNotSupported	The specified feature is not supported by the phone or network.
telErrGenericDrvNotFound	The generic driver could not be found.
telErrInvalidAppId	The specified application ID is not valid.
telErrInvalidDial	The dial string contains an invalid character.
telErrInvalidIndex	The index specified for accessing a value in storage is incorrect.
telErrInvalidParameter	A parameter is not valid.
telErrInvalidString	The text string contains an invalid character.

<b>Constant</b>	<b>Description</b>
telErrLibStillInUse	The shared lib is currently being used by another application. Do not unload it!
telErrMemAllocation	A memory allocation error occurred.
telErrMsgAllocation	The telephony messages pool is empty; a message could not be allocated.
telErrNetworkTimeOut	The network did not reply within the standard time delay amount.
telErrNoNetwork	There is no network available.
telErrNoSIMInserted	The SIM card is not inserted.
telErrNoSpecificDrv	A specific driver was not specified.
telErrNotInstalled	The shared library could not be installed.
telErrPassword	The password is not correct.
telErrPhoneCodeRequired	A phone code is required.
telErrPhoneComm	The communication link with the phone is down.
telErrPhoneMemAllocation	The phone's memory is full.
telErrPhoneMemFailure	The phone encountered a memory error.
telErrPhoneNumber	One of the following errors has occurred: the phone number is wrong, the SMS center is not valid, or the receiver phone number is wrong for the SMS.
telErrPhoneReply	The phone reply syntax is incorrect. Check the phone driver.
telErrPhoneToSIMPINRequired	A phone 2 SIM PIN code is required
telErrPIN2Required	A PIN2 code is required.
telErrPINRequired	A PIN code is required.
telErrPUK2Required	A PUK2 code is required.

## Telephony Basic Services

### Telephony Constants

---

Constant	Description
telErrPUKRequired	A PUK code is required.
telErrResultBusyResource	A resource is busy.
telErrResultTimeOut	A time-out was reached.
telErrResultUserCancel	The user cancelled the action.
telErrSecurity	Access to the phone was not granted.
telErrSettings	The telephony settings are not valid; this is due to 1) the Phone Panel preferences do not exist, or 2) the Telephony Profile is not correctly set.
telErrSIMBusy	The SIM could not reply.
telErrSIMFailure	The SIM is not working properly.
telErrSIMWrong	The SIM is not accepted by the phone.
telErrSpcCallError	The voice telephone call encountered an error.
telErrSpcLineIsBusy	The voice telephone call failed.
telErrSpcLineIsReleased	The voice telephone call has been released.
telErrSpecificDrvNotFound	The specified driver could not be found.
telErrTooManyApps	The applications table is full.
telErrTTaskNotFound	The Telephony Task could not be found.
telErrTTaskNotRunning	The Telephony Task is not running.
telErrUnavailableValue	The requested value can not be retrieved at the specified time. This is usually due to a TelSpcGetCallerNumber request when there is no active line.
telErrUnknown	An unknown telephony manager error occurred.
telErrValidityPeriod	The validity period specified for the short message is not valid.

Constant	Description
telErrValueStale	The information could not be retrieved; a copy of the most recently retrieved value has been returned instead.
telErrVersion	The shared library version does not match the version associated with the application.

## Information Type Constants

The [TelInfGetformattionType](#) structure uses the information type constants to encode the type of information to retrieve about the phone.

Constant	Value	Description
kTelInfPhoneBrand	0	The brand name of the phone.
kTelInfPhoneModel	1	The model number of the phone.
kTelInfPhoneRevision	2	The revision number of the phone.

## Keycode Constants

The [TelSndPlayKeyToneType](#) structure uses the keycode constants to specify the key tone to play.

Constant	Value	Description
kTel0Key	0x00	The 0 key on the phone keypad.
kTel1Key	0x01	The 1 key on the phone keypad.
kTel2Key	0x02	The 2 key on the phone keypad.
kTel3Key	0x03	The 3 key on the phone keypad.
kTel4Key	0x04	The 4 key on the phone keypad.
kTel5Key	0x05	The 5 key on the phone keypad.
kTel6Key	0x06	The 6 key on the phone keypad.
kTel7Key	0x07	The 7 key on the phone keypad.

## Telephony Basic Services

### Telephony Constants

---

Constant	Value	Description
kTel8Key	0x08	The 8 key on the phone keypad.
kTel9Key	0x09	The 9 key on the phone keypad.
kTelPoundKey	0x23	The POUND(#) key on the phone keypad.
kTelStarKey	0x2A	The STAR(*) key on the phone keypad.
kTelSendKey	0x45	The SEND key on the phone keypad.
kTelEndKey	0x46	The END key on the phone keypad.
kTelClrKey	0x47	The CLEAR key on the phone keypad.
kTelSaveKey	0x48	The SAVE key on the phone keypad.

## Key Sound Type Constants

The [TelSndPlayKeyType](#) structure uses the key sound type constants to specify how the tone is played.

Constant	Value	Description
kTelSndSingleTone	0x00	Play the key sound as a single tone.
kTelSndMultiTones	0x01	Play the key sound as a multiple tones.

## Message Identifier Constants

The message identifier constants are used with asynchronous calls to identify which telephony function is being or has been called. The TelMessages enumeration defines a constant for each function name.

Each message identifier constant has the form:

`kfunctionNameMessage`

where *functionName* is replaced by a function name.

The following table shows examples of message identifier constants. For a complete list, see the `TelephonyMgr.h` file.

Constant	Function
<code>kTelGetCallStateMessage</code>	<a href="#">TelGetCallState</a>
<code>kTelNwkSelectNetworkMessage</code>	<a href="#">TelNwkSelectNetwork</a>
<code>kTelSmsReadMessageMessage</code>	<a href="#">TelSmsReadMessage</a>

## Service Set Constants

The service set constants specify a set of API services.

Constant	Value	Description
<code>kTelNwkServiceId</code>	0	The network service set.
<code>kTelStyServiceId</code>	1	The security service set.
<code>kTelPowServiceId</code>	2	The power service set.
<code>kTelCfgServiceId</code>	3	The configuration service set.
<code>kTelSmsServiceId</code>	4	The short message service set.
<code>kTelEmcServiceId</code>	5	The emergency telephone call service set.
<code>kTelSpcServiceId</code>	6	The speech telephone call service set.
<code>kTelDtcServiceId</code>	7	The data telephone call service set.
<code>kTelPhbServiceId</code>	8	The phone book service set.
<code>kTelOemServiceId</code>	9	The OEM service set.
<code>kTelSndServiceId</code>	10	The sound service set.
<code>kTelInfServiceId</code>	11	The information service set.

## Telephony Functions

This section describes the functions used with the basic services portion of the telephony API.

## Telephony Basic Services

### Telephony Functions

---

## TelCancel

<b>Purpose</b>	Cancels an asynchronous function call.								
<b>Declared In</b>	TelephonyMgr.h								
<b>Prototype</b>	<pre>Err TelCancel(UINT16 iRefnum, TelAppID iAppId, UINT16 iTransId, UINT16 *ioTransIdP)</pre>								
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>-&gt; iTransId</td><td>The transaction ID associated with the function that you are cancelling.</td></tr><tr><td>&lt;-&gt; ioTransIdP</td><td><p>Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</p></td></tr></table>	-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	-> iTransId	The transaction ID associated with the function that you are cancelling.	<-> ioTransIdP	<p>Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</p>
-> iRefnum	The telephony manager library reference number.								
-> iAppId	The telephone application attachment identifier for your application.								
-> iTransId	The transaction ID associated with the function that you are cancelling.								
<-> ioTransIdP	<p>Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</p>								
<b>Synchronous Result</b>	Returns errNone if the function call was successfully cancelled. Returns the telErrCommandFailed error code if the function call could not be cancelled.								
<b>Asynchronous Result</b>	<p>The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:</p> <table><tr><td>returnCode</td><td>errNone upon success or an error code upon failure.</td></tr><tr><td>transId</td><td>The transaction ID of the operation.</td></tr><tr><td>functionId</td><td>kTelUrqCancelMessage</td></tr><tr><td>paramId</td><td>Points to the unsigned integer value passed to this function in the iTransId parameter.</td></tr></table>	returnCode	errNone upon success or an error code upon failure.	transId	The transaction ID of the operation.	functionId	kTelUrqCancelMessage	paramId	Points to the unsigned integer value passed to this function in the iTransId parameter.
returnCode	errNone upon success or an error code upon failure.								
transId	The transaction ID of the operation.								
functionId	kTelUrqCancelMessage								
paramId	Points to the unsigned integer value passed to this function in the iTransId parameter.								

<b>Comments</b>	This function cancels a pending asynchronous function call. You can cancel any asynchronous call except for an asynchronous call to the TelCancel function.  The function call that is cancelled returns the <code>telErrUserCancel</code> error code.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.

## TelClose

<b>Purpose</b>	Close the shared library.				
<b>Declared In</b>	<code>TelephonyMgr.h</code>				
<b>Prototype</b>	<code>Err TelClose(UINT16 iRefnum, TelAppID iAppId)</code>				
<b>Parameters</b>	<table><tr><td>-&gt; <code>iRefnum</code></td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; <code>iAppId</code></td><td>The telephone application attachment identifier for your application.</td></tr></table>	-> <code>iRefnum</code>	The telephony manager library reference number.	-> <code>iAppId</code>	The telephone application attachment identifier for your application.
-> <code>iRefnum</code>	The telephony manager library reference number.				
-> <code>iAppId</code>	The telephone application attachment identifier for your application.				
<b>Result</b>	Returns an error code, or error none if the library was successfully closed. If the library is currently being used by another application, this function returns the <code>telErrLibStillInUse</code> error code.				
<b>Comments</b>	<p>Call this function when you are done with the telephony manager. You can only use this function synchronously.</p> <p>If no other application is using the telephony manager, this function stops the Telephony task and releases any resources used by the telephony manager.</p>				
<b>See Also</b>	<a href="#">TelOpen</a>				

## Telephony Basic Services

### Telephony Functions

---

## TelClosePhoneConnection

**Purpose** Closes down communications with the connected phone.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelClosePhoneConnection(UInt16 iRefnum,  
TelAppID iAppId, UInt16 *ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> upon success or an error code upon failure.
<code>transId</code>	The transaction ID of the operation.
<code>paramP</code>	A NULL pointer.
<code>functionId</code>	<code>kTelUrqClosePhoneConnectionMessage</code>

**Comments** Call this function when you have finished communications with the phone and are ready to disconnect from it.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelOpenPhoneConnection](#)

## TelGetCallState

**Purpose** Retrieves the current telephone call state information.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelGetCallState(UInt16 iRefnum,  
TelAppID iAppId, TelGetCallStateType *ioParamP,  
UInt16 *ioTransIdP)`

<b>Parameters</b>	<p>-&gt; <code>iRefnum</code> The telephony manager library reference number.</p> <p>-&gt; <code>iAppId</code> The telephone application attachment identifier for your application.</p> <p>&lt;-&gt; <code>ioParamP</code> A pointer to a <a href="#">TelCallStateType</a> structure that describes the state of the current telephone call. On input, the <code>numberSize</code> field of this structure specifies the allocated size of the <code>number</code> buffer. Upon return, the <code>numberSize</code> field specifies the actual size of the telephone number, even if it was truncated to fit into the buffer.</p> <p>&lt;-&gt; <code>ioTransIdP</code> Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</p>
<b>Synchronous Result</b>	Returns <code>errNone</code> if the function was successful or returns an error code if not successful.

## Telephony Basic Services

### Telephony Functions

---

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the <a href="#">TelCallStateType</a> structure passed to this function in the <code>ioCallState</code> parameter.
functionId	<code>kTelGetCallStateMessage</code>

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

---

**Comments** This function retrieves information about the current telephone call state of the connection with the phone, and stores that information into the supplied [TelCallStateType](#) structure.

The current incoming or outgoing telephone call number is stored into the `number` field of the [TelCallStateType](#) structure referenced by `ioCallStateP`. If the `number` field buffer is too small to contain the complete telephone number, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `numberSize` field of the structure is always updated to contain the actual size of the complete telephone number.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSpcAcceptCall](#), [TelSpcCallNumber](#)

## TelGetEvent

**Purpose** Retrieves events for applications that use the telephony manager.

**Declared In** `TelephonyMgr.h`

**Prototype** `void TelGetEvent(UInt16 iRefnum, TelAppID iAppId,  
EventPtr oEventP, Int32 iTimeOut)`

**Parameters**

-> iRefnum	The telephony manager library reference number.
-> iAppId	The telephone application attachment identifier for your application.
<- oEventP	A pointer to a <a href="#">TelEventType</a> structure. Upon return, the structure contains the event information, which you should use as described in the Comments section.
-> iTimeout	Maximum number of ticks to wait before an event is returned ( <code>evtWaitForever</code> means wait indefinitely).

**Result** Returns nothing.

**Comments** This function retrieves both telephony and standard Palm OS® events. You must call this function to retrieve events in any application that is running in the UI task and using the telephony manager.

Upon return from this function, you need to test the type of the event by examining the `oEventP->type` field. If the event type is a telephony event, then you need to cast the pointer as follows to access the fields:

```
TelEventType *telEventP =  
    (TelEventType *)oEventP;
```

This function calls both the [EvtGetEvent](#) and [TelGetTelephonyEvent](#) functions to retrieve the next event for your application.

## Telephony Basic Services

### Telephony Functions

---

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [EvtGetEvent](#), [TelGetEvent](#)

## TelGetTelephonyEvent

**Purpose** Retrieves telephony events only.

**Declared In** `TelephonyMgr.h`

**Prototype** `void TelGetTelephonyEvent (UInt16 iRefnum,  
TelAppID iAppId, EventPtr oEventP,  
Int32 iTimeout)`

**Parameters**

-> iRefnum	The telephony manager library reference number.
-> iAppId	The telephone application attachment identifier for your application.
<- oEventP	A pointer to a <a href="#">TelEventType</a> structure. Upon return, the structure contains the event information.
-> iTimeout	Maximum number of ticks to wait before an event is returned (evtWaitForever means wait indefinitely).

**Result** Returns nothing.

**Comments** Use this function instead of the [TelGetEvent](#) function when you only want to process telephony events.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [EvtGetEvent](#), [TelGetEvent](#)

## TellInfGetInformation

<b>Purpose</b>	Retrieve brand, model, and revision information for the phone.								
<b>Declared In</b>	TelephonyMgr.h								
<b>Prototype</b>	<pre>Err TelInfGetInformation(UINT16 iRefnum,                            TelAppID iAppId,                            TelInfGetInformationType *ioParamP,                            UINT16 *ioTransIdP)</pre>								
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>&lt;-&gt; ioParamP</td><td>A pointer to a <a href="#">TelInfGetInformationType</a> structure.  On input, the <code>infoType</code> field of the structure contains the type of information that you want retrieved. The <code>size</code> field of this structure specifies the allocated size of the value buffer. Upon return, the <code>size</code> field specifies the actual size of the information that was retrieved, even if it was truncated to fit into the buffer.</td></tr><tr><td>&lt;-&gt; ioTransIdP</td><td>Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</td></tr></table>	-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	<-> ioParamP	A pointer to a <a href="#">TelInfGetInformationType</a> structure.  On input, the <code>infoType</code> field of the structure contains the type of information that you want retrieved. The <code>size</code> field of this structure specifies the allocated size of the value buffer. Upon return, the <code>size</code> field specifies the actual size of the information that was retrieved, even if it was truncated to fit into the buffer.	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
-> iRefnum	The telephony manager library reference number.								
-> iAppId	The telephone application attachment identifier for your application.								
<-> ioParamP	A pointer to a <a href="#">TelInfGetInformationType</a> structure.  On input, the <code>infoType</code> field of the structure contains the type of information that you want retrieved. The <code>size</code> field of this structure specifies the allocated size of the value buffer. Upon return, the <code>size</code> field specifies the actual size of the information that was retrieved, even if it was truncated to fit into the buffer.								
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.								
<b>Synchronous Result</b>	Returns <code>errNone</code> if the function was successful or returns an error code if not successful.								
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:								

## Telephony Basic Services

### Telephony Functions

---

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the <a href="#">TelInfGetInformationType</a> structure passed to this function in the ioInfoP parameter.
functionId	kTelInfGetInformationMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

---

<b>Comments</b>	<p>Call this function to retrieve information about the currently connected phone.</p> <p>The retrieved information is stored into the value field of the <a href="#">TelInfGetInformationType</a> referenced by <code>ioInfoP</code> structure. If the value field buffer is too small to contain the complete information, the value is truncated and this function returns the <code>telErrBufferSize</code> error. The <code>size</code> field of the structure is always updated to contain the actual size of the retrieved information.</p> <p>Before using this function, you should verify that it is available by calling the <a href="#">TelIsInfServiceAvailable</a> macro.</p>
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.

## TellsCfgServiceAvailable

<b>Purpose</b>	A macro that determines if the configuration service set is available in the current environment.	
<b>Declared In</b>	TelephonyMgr.h	
<b>Prototype</b>	TelIsCfgServiceAvailable (iRefnum, iAppId, ioTransIdP)	
<b>Parameters</b>	-> iRefnum	The telephony manager library reference number.
	-> iAppId	The telephone application attachment identifier for your application.
	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
<b>Synchronous Result</b>	Returns errNone if the service set is available, or an error code if not.	
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:  returnCode      errNone if the service set is available, or an error code if not.  transId          The transaction ID of the operation.  paramP          kTelCfgServiceId  functionId      kTelUrqIsServiceAvailableMessage	
<b>Comments</b>	You need to call this macro before calling any function in the configuration service set, which is the family of functions that begin with the TelCfg prefix.	

## Telephony Basic Services

### Telephony Functions

---

The configuration service set functions are documented in [Chapter 72, “Telephony SMS.”](#)

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelIs<FunctionName>Supported](#)

## TellsDtcServiceAvailable

**Purpose** A macro that determines if the data calls service set is available in the current environment.

**Declared In** `TelephonyMgr.h`

**Prototype** `TelIsDtcServiceAvailable (iRefnum, iAppId,  
ioTransIdP)`

**Parameters**

<code>-&gt; iRefnum</code>	The telephony manager library reference number.
<code>-&gt; iAppId</code>	The telephone application attachment identifier for your application.
<code>&lt;-&gt; ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the service set is available, or an error code if not.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> if the service set is available, or an error code if not.
<code>transId</code>	The transaction ID of the operation.

```
paramP      kTelDtcServiceId  
functionId   kTelUrqIsServiceAvailableMessage
```

**Comments** You need to call this macro before calling any function in the data calls service set, which is the family of functions that begin with the TelDtc prefix.  
The data calls service set functions are documented in [Chapter 71, “Telephony Calls.”](#)

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelIs<FunctionName>Supported](#)

## TellsEmcServiceAvailable

**Purpose** A macro that determines if the emergency calls service set is available in the current environment.

**Declared In** TelephonyMgr.h

**Prototype** TelIsEmcServiceAvailable (iRefnum, iAppId,  
ioTransIdP)

**Parameters**

-> iRefnum	The telephony manager library reference number.
-> iAppId	The telephone application attachment identifier for your application.
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the service set is available, or an error code if not.

## Telephony Basic Services

### Telephony Functions

---

<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:
returnCode	errNone if the service set is available, or an error code if not.
transId	The transaction ID of the operation.
paramP	kTelEmcServiceId
functionId	kTelUrqIsServiceAvailableMessage
<b>Comments</b>	You need to call this macro before calling any function in the emergency calls service set, which is the family of functions that begin with the TelEmc prefix.  The emergency calls service set functions are documented in <a href="#">Chapter 71, “Telephony Calls.”</a>
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelIs&lt;FunctionName&gt;Supported</a>

## TellsInfServiceAvailable

<b>Purpose</b>	A macro that determines if the information service set is available in the current environment.						
<b>Declared In</b>	TelephonyMgr.h						
<b>Prototype</b>	<code>TelIsInfServiceAvailable (iRefnum, iAppId, ioTransIdP)</code>						
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>&lt;-&gt; ioTransIdP</td><td>Set the value of this parameter to NULL to cause the function to execute synchronously.</td></tr></table>	-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.
-> iRefnum	The telephony manager library reference number.						
-> iAppId	The telephone application attachment identifier for your application.						
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.						

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the service set is available, or an error code if not.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode      errNone if the service set is available, or an error code if not.

transId          The transaction ID of the operation.

paramP          kTelInfServiceId

functionId      kTelUrqIsServiceAvailableMessage

**Comments** You need to call this macro before calling any function in the information service set, which is the family of functions that begin with the TelInf prefix.

The information service set functions are documented in this chapter.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelIs<FunctionName>Supported](#)

## Telephony Basic Services

### Telephony Functions

---

## TellsNwkServiceAvailable

**Purpose** A macro that determines if the network service set is available in the current environment.

**Declared In** `TelephonyMgr.h`

**Prototype** `TelIsNwkServiceAvailable (iRefnum, iAppId,  
ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iappId</code>	The telephone application attachment identifier for your application.
<code>&lt;-&gt; ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the service set is available, or an error code if not.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> if the service set is available, or an error code if not.
<code>transId</code>	The transaction ID of the operation.
<code>paramP</code>	<code>kTelNwkServiceId</code>
<code>functionId</code>	<code>kTelUrqIsServiceAvailableMessage</code>

**Comments** You need to call this macro before calling any function in the network service set, which is the family of functions that begin with the `TelNwk` prefix.

The network service set functions are documented in [Chapter 70, “Telephony Network.”](#)

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelIs<FunctionName>Supported](#)

## **TellsOemServiceAvailable**

**Purpose** A macro that determines if the OEM service set is available in the current environment.

**Declared In** `TelephonyMgr.h`

**Prototype** `TelIsOemServiceAvailable (iRefnum, iAppId,  
ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the service set is available, or an error code if not.

**Asynchronous Result** The following fields are updated in the [`TelEventType`](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> if the service set is available, or an error code if not.
<code>transId</code>	The transaction ID of the operation.

## Telephony Basic Services

### Telephony Functions

---

```
paramP      kTelOemServiceId  
functionId   kTelUrqIsServiceAvailableMessage
```

<b>Comments</b>	You need to call this macro before calling any function in the OEM service set, which is the family of functions that begin with the <code>TelOem</code> prefix.  The OEM service set functions are documented in this chapter.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">Tells&lt;FunctionName&gt;Supported</a>

## TellsPhbServiceAvailable

<b>Purpose</b>	A macro that determines if the phone book service set is available in the current environment.	
<b>Declared In</b>	<code>TelephonyMgr.h</code>	
<b>Prototype</b>	<code>TelIsPhbServiceAvailable (iRefnum, iAppId, ioTransIdP)</code>	
<b>Parameters</b>	-> <code>iRefnum</code>	The telephony manager library reference number.
	-> <code>iAppId</code>	The telephone application attachment identifier for your application.
	<-> <code>ioTransIdP</code>	Set the value of this parameter to <code>NULL</code> to cause the function to execute synchronously.  If this parameter is not <code>NULL</code> , the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
<b>Synchronous Result</b>	Returns <code>errNone</code> if the service set is available, or an error code if not.	

<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:
returnCode	errNone if the service set is available, or an error code if not.
transId	The transaction ID of the operation.
paramP	kTelPhbServiceId
functionId	kTelUrqIsServiceAvailableMessage
<b>Comments</b>	You need to call this macro before calling any function in the phone book service set, which is the family of functions that begin with the TelPhb prefix.  The phone book service set functions are documented in <a href="#">Chapter 73, “Telephony Phone Book.”</a>
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelIs&lt;FunctionName&gt;Supported</a>

## TellsPhoneConnected

<b>Purpose</b>	Determines if a phone is connected.						
<b>Declared In</b>	TelephonyMgr.h						
<b>Prototype</b>	Err TelIsPhoneConnected(UInt16 iRefnum, TelAppID iAppId, UInt16 *ioTransIdP)						
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>&lt;-&gt; ioTransIdP</td><td>Set the value of this parameter to NULL to cause the function to execute synchronously.</td></tr></table>	-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.
-> iRefnum	The telephony manager library reference number.						
-> iAppId	The telephone application attachment identifier for your application.						
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.						

## Telephony Basic Services

### Telephony Functions

---

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode      errNone upon success or an error code upon failure.

transId      The transaction ID of the operation.

paramP      A NULL pointer.

functionId      kTelUrqIsPhoneConnectedMessage

**Comments** Call this function to determine if there is currently a phone connected.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## TellsPowServiceAvailable

**Purpose** A macro that determines if the power services set is available in the current environment.

**Declared In** TelephonyMgr.h

**Prototype** TelIsPowServiceAvailable (iRefnum, iAppId,  
ioTransIdP)

**Parameters**

-> iRefnum	The telephony manager library reference number.
-> iAppId	The telephone application attachment identifier for your application.

<-> `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the service set is available, or an error code if not.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

`returnCode`      `errNone` if the service set is available, or an error code if not.

`transId`      The transaction ID of the operation.

`paramP`      `kTelPowServiceId`

`functionId`      `kTelUrqIsServiceAvailableMessage`

**Comments** You need to call this macro before calling any function in the power service set, which is the family of functions that begin with the `TelPow` prefix.

The power service set functions are documented in this chapter.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelIs<FunctionName>Supported](#)

## Telephony Basic Services

### Telephony Functions

---

## TellsSmsServiceAvailable

**Purpose** A macro that determines if the Short Message Service (SMS) service set is available in the current environment.

**Declared In** `TelephonyMgr.h`

**Prototype** `TelIsSmsServiceAvailable (iRefnum, iAppId,  
ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iappId</code>	The telephone application attachment identifier for your application.
<code>&lt;-&gt; ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the service set is available, or an error code if not.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> if the service set is available, or an error code if not.
<code>transId</code>	The transaction ID of the operation.
<code>paramP</code>	<code>kTelSmsServiceId</code>
<code>functionId</code>	<code>kTelUrqIsServiceAvailableMessage</code>

**Comments** You need to call this macro before calling any function in the SMS service set, which is the family of functions that begin with the `TelSms` prefix.

The SMS service set functions are documented in [Chapter 72](#), “[Telephony SMS](#).”

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelIs<FunctionName>Supported](#)

## **TellsSndServiceAvailable**

**Purpose** A macro that determines if the sound service set is available in the current environment.

**Declared In** `TelephonyMgr.h`

**Prototype** `TelIsSndServiceAvailable (iRefnum, iAppId,  
ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the service set is available, or an error code if not.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> if the service set is available, or an error code if not.
<code>transId</code>	The transaction ID of the operation.

## Telephony Basic Services

### Telephony Functions

---

```
paramP      kTelSndServiceId  
functionId   kTelUrqIsServiceAvailableMessage
```

**Comments** You need to call this macro before calling any function in the sound service set, which is the family of functions that begin with the TelSnd prefix.  
The sound service set functions are documented in this chapter.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelIs<FunctionName>Supported](#)

## TellsSpcServiceAvailable

**Purpose** A macro that determines if the speech telephone call service set is available in the current environment.

**Declared In** TelephonyMgr.h

**Prototype** TelIsSpcServiceAvailable (iRefnum, iAppId,  
ioTransIdP)

**Parameters**

-> iRefnum	The telephony manager library reference number.
-> iAppId	The telephone application attachment identifier for your application.
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the service set is available, or an error code if not.

<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:
returnCode	errNone if the service set is available, or an error code if not.
transId	The transaction ID of the operation.
paramP	kTelSpcServiceId
functionId	kTelUrqIsServiceAvailableMessage
<b>Comments</b>	You need to call this macro before calling any function in the speech telephone call service set, which is the family of functions that begin with the TelSpc prefix.  The speech telephone call service set functions are documented in <a href="#">Chapter 71, “Telephony Calls.”</a>
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelIs&lt;FunctionName&gt;Supported</a>

## TellsStyServiceAvailable

<b>Purpose</b>	A macro that determines if the security service set is available in the current environment.						
<b>Declared In</b>	TelephonyMgr.h						
<b>Prototype</b>	<code>TelIsStyServiceAvailable (iRefnum, iAppId,                                ioTransIdP)</code>						
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>&lt;-&gt; ioTransIdP</td><td>Set the value of this parameter to NULL to cause the function to execute synchronously.</td></tr></table>	-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.
-> iRefnum	The telephony manager library reference number.						
-> iAppId	The telephone application attachment identifier for your application.						
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.						

## Telephony Basic Services

### Telephony Functions

---

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

<b>Synchronous Result</b>	Returns errNone if the service set is available, or an error code if not.
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:  returnCode      errNone if the service set is available, or an error code if not.  transId          The transaction ID of the operation.  paramP          kTelStyServiceId  functionId      kTelUrqIsServiceAvailableMessage
<b>Comments</b>	You need to call this macro before calling any function in the security service set, which is the family of functions that begin with the TelSty prefix.  The security service set functions are documented in <a href="#">Chapter 70, "Telephony Network."</a>
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelIs&lt;FunctionName&gt;Supported</a>

## TelMatchPhoneDriver

<b>Purpose</b>	Determines if the currently selected driver matches the connected phone.								
<b>Declared In</b>	TelephonyMgr.h								
<b>Prototype</b>	<pre>Err TelMatchPhoneDriver(UInt16 iRefnum, TelAppID iAppId, UInt16 *ioTransIdP)</pre>								
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>&lt;-&gt; ioTransIdP</td><td><p>Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</p></td></tr></table>	-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	<-> ioTransIdP	<p>Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</p>		
-> iRefnum	The telephony manager library reference number.								
-> iAppId	The telephone application attachment identifier for your application.								
<-> ioTransIdP	<p>Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</p>								
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.								
<b>Asynchronous Result</b>	<p>The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:</p> <table><tr><td>returnCode</td><td>errNone upon success or an error code upon failure</td></tr><tr><td>transId</td><td>The transaction ID of the operation.</td></tr><tr><td>paramP</td><td>A NULL pointer.</td></tr><tr><td>functionId</td><td>kTelUrgMatchPhoneDriverMessage</td></tr></table>	returnCode	errNone upon success or an error code upon failure	transId	The transaction ID of the operation.	paramP	A NULL pointer.	functionId	kTelUrgMatchPhoneDriverMessage
returnCode	errNone upon success or an error code upon failure								
transId	The transaction ID of the operation.								
paramP	A NULL pointer.								
functionId	kTelUrgMatchPhoneDriverMessage								
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.								

## Telephony Basic Services

### Telephony Functions

---

## TelOemCall

**Purpose** Pass a call to an OEM function.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelOemCall(UInt16 iRefnum, TelAppID iAppId,  
TelOemCallType *ioParamP, UInt16 *ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
<-> <code>ioParamP</code>	A pointer to a <a href="#">TelOemCallType</a> structure that contains information about the OEM function call.
<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> upon success or an error code upon failure.
<code>transID</code>	matches the output value of the <code>ioTransIdP</code> parameter

functionID	matches the function ID in the <a href="#">TelOemCallType</a> structure passed to this function in the <code>ioParamP</code> parameter
paramId	points to the <a href="#">TelOemCallType</a> structure passed to this function in the <code>ioParamP</code> parameter

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

---

**Comments**

Call this function to send a request to an OEM function. The calling function and the OEM function are responsible for coordinating the parameter block that is passed in the [TelOemCallType](#) structure. Before using this function, you should verify that it is available by calling the [TelIsOemServiceAvailable](#) macro.

**Compatibility**

Implemented only if [4.0 New Feature Set](#) is present.

## TelOpen

**Purpose**

Open the telephony manager API to initialize telephony services and launch the telephony task.

**Declared In**

`TelephonyMgr.h`

**Prototype**

```
Err TelOpen(UInt16 iRefnum, UInt32 iVersnum,  
TelAppID *oAppIdP)
```

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iVersnum</code>	The version number of the shared library for which your application was developed.

## Telephony Basic Services

### Telephony Functions

---

<- oAppIdP      A pointer to an application ID value. Upon return, this is the application ID that you supply as a parameter to the any other telephony functions that you call.

**Result**      Returns errNone if the function was successful or returns an error code if not successful. The following errors can occur:

- the telephony task could not be found  
(telErrTTaskNotFound)
- the telephony task could not be launched  
(telErrTTaskNotRunning)
- the phone driver could not be found
- the shared library version is not valid

**Comments**      You can only call this function synchronously. You must call this function before calling any other telephony manager functions.

You can specify the current version of the shared library by using the kTelMgrVersion constant as the value of the iVersnum parameter.

**Compatibility**      Implemented only if [4.0 New Feature Set](#) is present.

**See Also**      [TelClose](#)

## TelOpenPhoneConnection

**Purpose**      Open communications with the connected phone.

**Declared In**      TelephonyMgr.h

**Prototype**      Err TelOpenPhoneConnection (UInt16 iRefnum,  
TelAppID iAppId, UInt16 \*ioTransIdP)

**Parameters**

-> iRefnum	The telephony manager library reference number.
-> iAppId	The telephone application attachment identifier for your application.

<-> `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

`returnCode`      `errNone` upon success or an error code upon failure.

`transId`      The transaction ID of the operation.

`paramP`      A NULL pointer.

`functionId`      `kTelUrqOpenPhoneConnectionMessage`

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelClosePhoneConnection](#)

## TelPowGetBatteryStatus

**Purpose** Retrieves the status of the phone's battery.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelPowGetBatteryStatus (UInt16 iRefnum,  
TelAppID iAppId, UInt8 *oStatusP,  
UInt16 *ioTransIdP)`

**Parameters** `-> iRefnum` The telephony manager library reference number.

`-> iAppId` The telephone application attachment identifier for your application.

## Telephony Basic Services

### Telephony Functions

---

<- oStatusP	A pointer to an unsigned byte value. Upon return, this is the battery status value, which is one of the <a href="#">Battery Status Constants</a> .
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the unsigned integer value passed to this function in the oStatusP parameter.
functionId	kTelPowBatteryStatusMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by oStatusP remains in memory until the asynchronous call completes.

---

**Comments** Before using this function, you should verify that it is available by calling the [TelIsPowServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelPowGetPowerLevel](#)

## TelPowGetPowerLevel

<b>Purpose</b>	Retrieve the current level of the phone battery, as a percentage value.	
<b>Declared In</b>	TelephonyMgr.h	
<b>Prototype</b>	<pre>Err TelPowGetPowerLevel(UInt16 iRefnum, TelAppID iAppId, UInt8 *oPowerP, UInt16 *ioTransIdP)</pre>	
<b>Parameters</b>	-> iRefnum	The telephony manager library reference number.
	-> iAppId	The telephone application attachment identifier for your application.
	<- oPowerP	A pointer to an unsigned byte value. Upon return, this is the battery percentage value.
	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.	
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:	
	returnCode	errNone upon success or an error code upon failure.
	transId	The transaction ID of the operation.
	paramP	Points to the unsigned integer value passed to this function in the oPowerP parameter.
	functionId	kTelPowGetPowerLevelMessage

## Telephony Basic Services

### Telephony Functions

---

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by `oPowerP` remains in memory until the asynchronous call completes.

---

<b>Comments</b>	The returned percentage value is an integer value between 0 and 100.  Before using this function, you should verify that it is available by calling the <a href="#">TelIsPowServiceAvailable</a> macro.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelPowGetBatteryStatus</a>

## TelPowSetPhonePower

<b>Purpose</b>	Turns the phone on or off.						
<b>Declared In</b>	TelephonyMgr.h						
<b>Prototype</b>	<code>Err TelPowSetPhonePower(UInt16 iRefnum, TelAppID iAppId, Boolean iPowerOn)</code>						
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>-&gt; iPowerOn</td><td>Set to <code>true</code> to turn the phone on, and set to <code>false</code> to turn the phone off.</td></tr></table>	-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	-> iPowerOn	Set to <code>true</code> to turn the phone on, and set to <code>false</code> to turn the phone off.
-> iRefnum	The telephony manager library reference number.						
-> iAppId	The telephone application attachment identifier for your application.						
-> iPowerOn	Set to <code>true</code> to turn the phone on, and set to <code>false</code> to turn the phone off.						
<b>Result</b>	Returns <code>errNone</code> if the function was successful and an error code if not.						
<b>Comments</b>	This function can only be called synchronously.  Before using this function, you should verify that it is available by calling the <a href="#">TelIsPowServiceAvailable</a> macro.						

This function corresponds to the kTelPowSetPhonePowerMessage function ID value.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## TelSendCommandString

**Purpose** Sends a command string to the phone or to the network.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelSendCommandString(UInt16 iRefnum,  
TelAppID iappId,  
TelSendCommandStringType *ioParamP,  
UInt16 *ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iappId</code>	The telephone application attachment identifier for your application.
<-> <code>ioParamP</code>	A pointer to a command string structure of type <a href="#">TelSendCommandStringType</a> .
<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

## Telephony Basic Services

### Telephony Functions

---

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the <a href="#">TelSendCommandStringType</a> structure passed to this function in the <code>ioParam</code> parameter.
functionId	Matches the function ID in the <a href="#">TelOemCallType</a> referenced by the <code>ioParamP</code> structure.

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

---

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## TelSndMute

**Purpose** Mute or un-mute an active telephone call.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelSndMute(UINT16 iRefnum, TelAppID iAppId,  
Boolean iMuteOn, UINT16 *ioTransIdP)`

<b>Parameters</b>	-> <code>iRefnum</code>	The telephony manager library reference number.
	-> <code>iAppId</code>	The telephone application attachment identifier for your application.
	-> <code>iMuteOn</code>	Set to <code>true</code> to mute the telephone call, or set to <code>false</code> to unmute the telephone call.
	<-> <code>ioTransIdP</code>	Set the value of this parameter to <code>NULL</code> to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode      errNone upon success or an error code upon failure.

transId      The transaction ID of the operation.

paramP      Points to the Boolean value passed to this function in the iMuteOn parameter.

functionID      kTelSndMuteMessage

**Comments** Before using this function, you should verify that it is available by calling the [TelIsSndServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSndPlayKeyTone](#)

## TelSndPlayKeyTone

**Purpose** Play a keytone sound on the phone.

**Declared In** TelephonyMgr.h

**Prototype** Err TelSndPlayKeyTone(UINT16 iRefnum,  
                  TelAppID iAppId, TelSndPlayKeyToneType \*iParamP,  
                  UINT16 \*ioTransIdP)

**Parameters** -> iRefnum      The telephony manager library reference number.

## Telephony Basic Services

### Telephony Functions

---

-> iAppId	The telephone application attachment identifier for your application.
-> iParamP	A pointer to a <a href="#">TelSndPlayKeyToneType</a> structure that describes the tone to play.
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the <a href="#">TelSndPlayKeyToneType</a> structure passed to this function in the iKeyToneP parameter.
functionId	kTelSndPlayKeyTone

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by iParamP remains in memory until the asynchronous call completes.

---

**Comments** Before using this function, you should verify that it is available by calling the [TelIsSndServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSndStopKeyTone](#)

## TelSndStopKeyTone

<b>Purpose</b>	Stop the playing of a keytone sound on the phone.								
<b>Declared In</b>	TelephonyMgr.h								
<b>Prototype</b>	<pre>Err TelSndStopKeyTone(UINT16 iRefnum, TelAppID iAppId, UINT16 *ioTransIdP)</pre>								
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>&lt;-&gt; ioTransIdP</td><td>Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</td></tr></table>	-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.		
-> iRefnum	The telephony manager library reference number.								
-> iAppId	The telephone application attachment identifier for your application.								
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.								
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.								
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes: <table><tr><td>returnCode</td><td>errNone upon success or an error code upon failure.</td></tr><tr><td>transId</td><td>The transaction ID of the operation.</td></tr><tr><td>paramP</td><td>A NULL pointer.</td></tr><tr><td>functionId</td><td>kTelSndStopKeyToneMessage</td></tr></table>	returnCode	errNone upon success or an error code upon failure.	transId	The transaction ID of the operation.	paramP	A NULL pointer.	functionId	kTelSndStopKeyToneMessage
returnCode	errNone upon success or an error code upon failure.								
transId	The transaction ID of the operation.								
paramP	A NULL pointer.								
functionId	kTelSndStopKeyToneMessage								
<b>Comments</b>	Before using this function, you should verify that it is available by calling the <a href="#">TelIsSndServiceAvailable</a> macro.								

## Telephony Basic Services

### Feature Support Functions

---

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSndPlayKeyTone](#)

## Feature Support Functions

This section describes the functions that you can call to determine if a specific feature or function is supported in the current operating environment.

### Tells<FunctionName>Supported

**Purpose** Determines if the specified function is supported.

**Declared In** `TelephonyMgr.h`

**Prototype** `TelIs<FunctionName>Supported (iRefnum, iAppId,  
ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
<code>&lt;-&gt; ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the specified function is supported.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode	errNone if the specified function is supported, or an error code if not.
transId	The transaction ID of the operation.
paramP	The function ID of the function for which you are testing. For example, if you call <code>TelIsCgfGetSmsCenterSupported</code> , the value of this field is <code>kTelCfgGetSmsCenterMessage</code> .
functionId	The function ID value for each function is described in the documentation for the function.
	<code>kTelUrqIsFunctionSupportedMessage</code>

## Comments

This is a family of synchronous macros that test if a specific function is available in the current environment.

To use the macro, substitute a function name for the `<FunctionName>` portion of the macro name. You can substitute any Telephony Manager function name; for a complete list of the Telephony Manager functions, see “[Summary of Telephony Manager](#)” on page 235 in *Palm OS Programmer’s Companion*, vol. II, *Communications*.

For example, to determine if the [`TelNwkGetSignalLevel`](#) function is available in the current environment, call the `TelIsNwkGetSignalLevelSupported` macro.

---

**NOTE:** A service set can be available without all of its functions being available. Thus, if the [`TelIs<ServiceSet>Available`](#) macro returns `true` for a specific service set, you know that the service set is available, but you need to call [`TelIs<FunctionName>Supported`](#) to determine if a specific function is available.

---

This macro corresponds to the `kTelUrqIsFunctionSupportedMessage` function ID value.

## Telephony Basic Services

### Feature Support Functions

---

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelIs<ServiceSet>Available](#)

## Tells<ServiceSet>Available

**Purpose** Determines if the specified service set is available.

**Declared In** `TelephonyMgr.h`

**Prototype** `TelIs<ServiceSet>Available (iRefnum, iAppId,  
ioTransIdP)`

**Parameters**

<code>-&gt; iRefnum</code>	The telephony manager library reference number.
<code>-&gt; iappId</code>	The telephone application attachment identifier for your application.
<code>&lt;-&gt; ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the service set is available.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> if the service set is available, or an error code if not.
<code>transId</code>	The transaction ID of the operation.

paramP      The service ID of the service set for which you are testing. For example, if you call `TelIsCfgServiceAvailable`, the value of this field is `kTelCfgServiceId`.

The service IDs are described in [Service Set Constants](#).

functionId    `kTelUrqIsServiceAvailableMessage`

**Comments**    This is a family of synchronous macros that test if a specific service set is available. You must call the appropriate set availability function before calling a function in the set.

---

**NOTE:** A service set can be available without all of its functions being available. You can use this macro to determine the availability of a specific service set, which you might use to determine the configuration of your applications' user interface. To test if a specific function is supported, use the [`TelIs<FunctionName>Supported`](#) macro.

---

You can call these specific macros to determine if the service set is available:

- [`TelIsCfgServiceAvailable`](#) to determine if the configuration service set is available.
- [`TelIsDtcServiceAvailable`](#) to determine if the data calls service set is available.
- [`TelIsEmcServiceAvailable`](#) to determine if the emergency calls service set is available.
- [`TelIsInfServiceAvailable`](#) to determine if the information service set is available.
- [`TelIsNwkServiceAvailable`](#) to determine if the network service set is available.
- [`TelIsOemServiceAvailable`](#) to determine if the OEM service set is available.
- [`TelIsPhbServiceAvailable`](#) to determine if the phone book service set is available.

## Telephony Basic Services

### Feature Support Functions

---

- [TelIsPowServiceAvailable](#) to determine if the power service set is available.
- [TelIsSmsServiceAvailable](#) to determine if the SMS service set is available.
- [TelIsSndServiceAvailable](#) to determine if the sound service set is available.
- [TelIsSpcServiceAvailable](#) to determine if the speech calls service set is available.
- [TelIsStyServiceAvailable](#) to determine if the security service set is available.

Each of these macros corresponds to the `kTelUrgIsServiceSupportedMessage` function ID value.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelIs<FunctionName>Supported](#)

# Telephony Security and Configuration

---

This chapter describes the telephony security and configuration service sets of the telephony API.

For more information about the telephony manager basic services and the different service sets, see [Chapter 68, “Telephony Basic Services.”](#)

This chapter describes:

- [Telephony Security and Configuration Data Structures](#)
- [Telephony Security and Configuration Constants](#)
- [Telephony Security and Configuration Functions](#)

For more information about using the telephony manager, see [Chapter 10, “Telephony Manager”](#) in *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## Telephony Security and Configuration Data Structures

This section describes the data structures used with the telephony security and configuration service sets portion of the telephony API.

### TelCfgGetPhoneNumberType

The [TelCfgGetPhoneNumber](#) function uses a `TelCfgGetPhoneNumberType` structure to retrieve the connected phone dial number.

```
typedef struct _TelCfgGetPhoneNumberType
{
    UInt8      size;
    Char*      value;
} TelCfgGetPhoneNumberType
```

### Field Descriptions

<-> size	The size of the value buffer.  When the structure is used as an input parameter, this is the allocated size, in bytes, of the value buffer.  Upon return, this is the actual size of the string, including the null terminator character. If the value buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a <code>telErrBufferSize</code> error.
<- value	A buffer into which the dial number is stored.  Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a <code>telErrBufferSize</code> error.

## TelCfgGetSmsCenterType

The [TelCfgGetSmsCenter](#) function uses a `TelCfgGetSmsCenterType` structure to retrieve the SMS service center dial number.

```
typedef struct _TelCfgGetSmsCenterType
{
    UInt8      size;
    Char*      value;
} TelCfgGetSmsCenterType
```

### Field Descriptions

<-> size

The size of the value buffer.

When the structure is used as an input parameter, this is the allocated size, in bytes, of the value buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the value buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

<- value

A buffer into which the dial number is stored.

Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a `telErrBufferSize` error.

## TelStyChangeAuthenticationType

You use the `TelStyChangeAuthenticationType` to change an authentication code with the [TelStyChangeAuthenticationCode](#) function.

```
typedef struct _TelStyChangeAuthenticationType
{
    UInt      codeId;
    Char*    oldCode;
    Char*    newCode;
} TelStyChangeAuthenticationType
```

### Field Descriptions

-> codeId

The ID of the authentication code to change.

-> oldCode

The previous value of the code.

-> newCode

The new value of the code.

## **Telephony Security and Configuration**

*Telephony Security and Configuration Constants*

---

# **Telephony Security and Configuration Constants**

This section describes the constants used with the telephony security and configuration service sets of the telephony API.

## **Authentication State Constants**

The authentication state constants describe the current authentication state of the mobile unit connection.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
kTelStyReady	0	No additional security information is expected.
kTelStyPin1CodeId	1	The PIN1 code is expected.
kTelStyPin2CodeId	2	The PIN2 code is expected.
kTelStyPuk1CodeId	3	The PUK1 code is expected.
kTelStyPuk2CodeId	4	The PUK2 code is expected.
kTelStyPhoneToSimCodeId	5	The phone-to-SIM code is expected.
kTelStyFirstOemCodeId	6	An OEM code is expected.
		The constant kTelStyFirstOemCodeId specifies the first OEM authentication code. You can specify additional OEM codes by incrementing this value. For example, to specify the third OEM authentication code, use the following: kTelStyFirstOemCodeId+2

## **Telephony Security and Configuration Functions**

This section describes the data structures used with the telephony security and configuration service sets portion of the telephony API.

## TelCfgGetPhoneNumber

<b>Purpose</b>	Retrieve the connected telephone number.									
<b>Declared In</b>	TelephonyMgr.h									
<b>Prototype</b>	<pre>Err TelCfgGetPhoneNumber(UInt16 iRefnum, TelAppID iAppId, TelCfgGetPhoneNumberType* ioParamP, UInt16* ioTransIdP)</pre>									
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>&lt;-&gt; ioParamP</td><td>A pointer to a <a href="#">TelCfgGetPhoneNumberType</a> structure that is filled in with the dial telephone number.</td></tr><tr><td>&lt;-&gt; ioTransIdP</td><td>Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</td></tr></table>		-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	<-> ioParamP	A pointer to a <a href="#">TelCfgGetPhoneNumberType</a> structure that is filled in with the dial telephone number.	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
-> iRefnum	The telephony manager library reference number.									
-> iAppId	The telephone application attachment identifier for your application.									
<-> ioParamP	A pointer to a <a href="#">TelCfgGetPhoneNumberType</a> structure that is filled in with the dial telephone number.									
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.									
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.									
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes: <table><tr><td>returnCode</td><td>errNone upon success or an error code upon failure.</td></tr><tr><td>transId</td><td>The transaction ID of the operation.</td></tr></table>		returnCode	errNone upon success or an error code upon failure.	transId	The transaction ID of the operation.				
returnCode	errNone upon success or an error code upon failure.									
transId	The transaction ID of the operation.									

## Telephony Security and Configuration

### Telephony Security and Configuration Functions

---

paramP              Points to the [TelCfgGetPhoneNumberType](#) structure passed to this function in the ioParamP parameter.

functionId          kTelCfgGetPhoneNumberMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by ioParamP remains in memory until the asynchronous call completes.

---

**Comments**    The connected dial telephone number is stored into the value field of the [TelCfgGetPhoneNumberType](#) structure referenced by ioParamP. If the value field buffer is too small to contain the complete telephone number, the string is truncated (and ends with the null terminator character) and this function returns the telErrBufferSize error. The size field of the structure is always updated to contain the actual size of the complete telephone number.

Before using this function, you should verify that it is available by calling the [TelIsCfgServiceAvailable](#) macro.

**Compatibility**   Implemented only if [4.0 New Feature Set](#) is present.

**See Also**        [TelCfgSetSmsCenter](#), [TelSmsSendMessage](#)

## TelCfgGetSmsCenter

<b>Purpose</b>	Retrieve the SMS service center dial telephone number.									
<b>Declared In</b>	TelephonyMgr.h									
<b>Prototype</b>	<pre>Err TelCfgGetSmsCenter(UInt16 iRefnum, TelAppID iAppId, TelCfgGetSmsCenterType* ioParamP, UInt16* ioTransIdP)</pre>									
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>&lt;-&gt; ioParamP</td><td>A pointer to a <a href="#">TelCfgGetSmsCenterType</a> structure that is filled in with the dial telephone number.</td></tr><tr><td>&lt;-&gt; ioTransIdP</td><td>Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</td></tr></table>		-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	<-> ioParamP	A pointer to a <a href="#">TelCfgGetSmsCenterType</a> structure that is filled in with the dial telephone number.	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
-> iRefnum	The telephony manager library reference number.									
-> iAppId	The telephone application attachment identifier for your application.									
<-> ioParamP	A pointer to a <a href="#">TelCfgGetSmsCenterType</a> structure that is filled in with the dial telephone number.									
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.									
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.									
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes: <table><tr><td>returnCode</td><td>errNone upon success or an error code upon failure.</td></tr><tr><td>transId</td><td>The transaction ID of the operation.</td></tr></table>		returnCode	errNone upon success or an error code upon failure.	transId	The transaction ID of the operation.				
returnCode	errNone upon success or an error code upon failure.									
transId	The transaction ID of the operation.									

## Telephony Security and Configuration

### Telephony Security and Configuration Functions

---

paramP	Points to the <a href="#">TelCfgGetSmsCenterType</a> structure passed to this function in the ioParamP parameter.
functionId	kTelCfgGetSmsCenterMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

---

<b>Comments</b>	The service center dial telephone number is stored into the <code>value</code> field of the <a href="#">TelCfgGetSmsCenterType</a> structure referenced by <code>ioParamP</code> . If the <code>value</code> field buffer is too small to contain the complete telephone number, the string is truncated (and ends with the null terminator character) and this function returns the <code>telErrBufferSize</code> error. The <code>size</code> field of the structure is always updated to contain the actual size of the complete telephone number.  Before using this function, you should verify that it is available by calling the <a href="#">TelIsCfgServiceAvailable</a> macro.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelCfgSetSmsCenter</a> , <a href="#">TelSmsSendMessage</a>

## TelCfgSetSmsCenter

<b>Purpose</b>	Set the SMS service center dial telephone number.	
<b>Declared In</b>	<code>TelephonyMgr.h</code>	
<b>Prototype</b>	<code>Err TelCfgSetSmsCenter(UInt16 iRefnum, TelAppID iAppId, const Char* iDialNumberP, UInt16* ioTransIdP)</code>	
<b>Parameters</b>	<code>-&gt; iRefnum</code>	The telephony manager library reference number.

-> iAppId	The telephone application attachment identifier for your application.
-> iDialNumberP	A pointer to the null-terminated dial telephone number string for the SMS service center.
<-> ioTransIDP	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the string passed to this function in the iDialNumberP parameter.
functionId	kTelCfgSetSmsCenterMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by iDialNumberP remains in memory until the asynchronous call completes.

---

**Comments** Before using this function, you should verify that it is available by calling the [TelIsCfgServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelCfgGetSmsCenter](#), [TelSmsSendMessage](#)

## Telephony Security and Configuration

### Telephony Security and Configuration Functions

---

## TelStyChangeAuthenticationCode

**Purpose** Change the value of an authentication code. Note that you can only use this function with GSM networks.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelStyChangeAuthenticationCode (UInt16 iRefnum,  
TelAppID iAppId,  
TelStyChangeAuthenticationType* iParamP,  
UInt16* ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
-> <code>iParamP</code>	A pointer to a <a href="#">TelStyChangeAuthenticationType</a> structure that contains the old and new authentication code values.
<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> upon success or an error code upon failure.
<code>transId</code>	The transaction ID of the operation.

paramP	Points to the <a href="#">TelStyChangeAuthenticationType</a> structure passed to this function in the iParamP parameter.
functionId	kTelStyChangeAuthenticationCodeMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by iParamP remains in memory until the asynchronous call completes.

---

**Comments** Before using this function, you should verify that it is available by calling the [TelIsStyServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelStyEnterAuthenticationCode](#)

## TelStyEnterAuthenticationCode

**Purpose** Enter the authentication code for which the phone is currently waiting. Note that you can only use this function with GSM networks.

**Declared In** TelephonyMgr.h

**Prototype** Err TelStyEnterAuthenticationCode (UInt16 iRefnum,  
TelAppID iAppId, const Char\* iCodeP,  
UInt16\* ioTransIdP)

**Parameters**

-> iRefnum	The telephony manager library reference number.
-> iAppId	The telephone application attachment identifier for your application.
-> iCodeP	A pointer to the null-terminated authentication code string to send to the phone.

## Telephony Security and Configuration

### Telephony Security and Configuration Functions

---

<-> `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> upon success or an error code upon failure.
<code>transId</code>	The transaction ID of the operation.
<code>paramP</code>	Points to the string passed to this function in the <code>iCodeP</code> parameter.
<code>functionId</code>	<code>kTelStyEnterAuthenticationCodeMessage</code>

---

**WARNING!** When using this function asynchronously, you must ensure that the string referenced by `iCodeP` remains in memory until the asynchronous call completes.

---

**Comments** Before using this function, you should verify that it is available by calling the [TelIsStyServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelStyChangeAuthenticationCode](#),  
[TelStyGetAuthenticationState](#)

## TelStyGetAuthenticationState

<b>Purpose</b>	Returns the current state of user authentication.								
<b>Declared In</b>	TelephonyMgr.h								
<b>Prototype</b>	<pre>Err TelStyGetAuthenticationState(UInt16 iRefnum,                                   TelAppID iAppId, UInt8* oStateP,                                   UInt16* ioTransIdP)</pre>								
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>&lt;- oStateP</td><td>A pointer to an unsigned byte value. Upon return, this is the authentication state, which is one of the <a href="#">Authentication State Constants</a>.</td></tr><tr><td>&lt;-&gt; ioTransIdP</td><td>Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</td></tr></table>	-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	<- oStateP	A pointer to an unsigned byte value. Upon return, this is the authentication state, which is one of the <a href="#">Authentication State Constants</a> .	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
-> iRefnum	The telephony manager library reference number.								
-> iAppId	The telephone application attachment identifier for your application.								
<- oStateP	A pointer to an unsigned byte value. Upon return, this is the authentication state, which is one of the <a href="#">Authentication State Constants</a> .								
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.								
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.								
<b>Asynchronous Result</b>	<p>The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:</p> <table><tr><td>returnCode</td><td>errNone upon success or an error code upon failure.</td></tr><tr><td>transId</td><td>The transaction ID of the operation.</td></tr><tr><td>paramP</td><td>Points to the unsigned integer value passed to this function in the oStateP parameter.</td></tr><tr><td>functionId</td><td>kTelStyGetAuthenticationStateMessage</td></tr></table>	returnCode	errNone upon success or an error code upon failure.	transId	The transaction ID of the operation.	paramP	Points to the unsigned integer value passed to this function in the oStateP parameter.	functionId	kTelStyGetAuthenticationStateMessage
returnCode	errNone upon success or an error code upon failure.								
transId	The transaction ID of the operation.								
paramP	Points to the unsigned integer value passed to this function in the oStateP parameter.								
functionId	kTelStyGetAuthenticationStateMessage								

## Telephony Security and Configuration

### *Telephony Security and Configuration Functions*

---

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by `oStateP` remains in memory until the asynchronous call completes.

---

**Comments** Before using this function, you should verify that it is available by calling the [TelIsStyServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelStyEnterAuthenticationCode](#)

# Telephony Network

---

This chapter describes the telephony network service set of the telephony API.

For more information about the telephony manager basic services and the different service sets, see [Chapter 68, “Telephony Basic Services.”](#)

This chapter describes:

- [Telephony Network Data Structures](#)
- [Telephony Network Constants](#)
- [Telephony Network Functions](#)

For more information about using the telephony manager, see [Chapter 10, “Telephony Manager”](#) in *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## Telephony Network Data Structures

This section describes the data structures used with the telephony network service set portion of the telephony API.

### TelNwkGetLocationType

You use the TelNwkGetLocationType structure with the [TelNwkGetLocation](#) function to retrieve information about the location of the phone.

```
typedef struct _TelNwkGetLocationType
{
    Char*      value;
    UInt16     size;
} TelNwkGetLocationType
```

## Telephony Network

### Telephony Network Data Structures

---

#### Field Descriptions

- |          |   |
|----------|---|
| <- value | A buffer into which the current location string is stored. The format of this string is network-dependent.<br><br>Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a <code>telErrBufferSize</code> error.   |
| <-> size | The size of the <code>value</code> string. When the structure is used as an input parameter, this is the allocated size, in bytes, of the <code>value</code> buffer.<br><br>Upon return, this is the actual size of the string, including the null terminator character. If the <code>value</code> buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a <code>telErrBufferSize</code> error. |

### TelNwkGetNetworkNameType

You use the `TelNwkGetNetworkNameType` structure with the [TelNwkGetNetworkName](#) function to retrieve the name of the specified, registered network.

```
typedef struct _TelNwkGetNetworkNameType
{
    UInt32    id;
    Char*    value;
    UInt16    size;
} TelNwkGetNetworkNameType
```

### Field Descriptions

->	<code>id</code>	The network ID.
<-	<code>value</code>	A null-terminated string buffer into which the network name is stored.  Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a <code>telErrBufferSize</code> error.
<->	<code>size</code>	The size of the <code>value</code> string. When the structure is used as an input parameter, this is the allocated length, in bytes, of the <code>value</code> buffer.
		Upon return, this is the actual size of the string, including the null terminator character. If the <code>value</code> buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a <code>telErrBufferSize</code> error

## TelNwkGetNetworksType

You use the `TelNwkGetNetworksType` structure with the [`TelNwkGetNetworks`](#) function to retrieve the number of registered networks.

```
typedef struct _TelNwkGetNetworksType
{
    UInt32*   networkIdP;
    UInt8     size;
} TelNwkGetNetworksType
```

## Telephony Network

### Telephony Network Constants

---

#### Field Descriptions

<- networkIdP	An array into which the retrieved network IDs are stored.  Note that if this buffer is too small to contain all of the available IDs, the data is truncated and the function using this structure generates a <code>telErrBufferSize</code> error.
<-> size	When the structure is use as an input value, this is the allocated size, in elements, of the <code>networkIdP</code> array.
	When the structure is used as a return value, this is the number of network IDs that are available. If the <code>networkP</code> buffer is too small to contain all of the retrieved IDs, this field is assigned the entire number of available IDs, and the function using this structure generates a <code>telErrBufferSize</code> error

## Telephony Network Constants

This section describes the constants used with the telephony network service set of the telephony API, which include the following constant types:

- [Network Type Constants](#)
- [Network Search Mode Constants](#)

### Network Type Constants

The network type constants describe the type of network connection.

Constant	Value	Description
kTelNwkCDMA	0	A CDMA network.
kTelNwkGSM	1	A GSM network.

Constant	Value	Description
kTelNwkTDMA	2	A TDMA network.
kTelNwkPDC	3	A PDC network.

## Network Search Mode Constants

The network search mode constants describe the search mode used to find a network.

Constant	Value	Description
kTelNwkManualSearch	0	Manual network searching.
kTelNwkAutomaticSearch	1	Automatic network searching.

# Telephony Network Functions

This section describes the data structures used with the telephony network service set portion of the telephony API.

## TelNwkGetLocation

**Purpose** Retrieve information about the location of the mobile unit.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelNwkGetLocation(UINT16 iRefnum,  
 TelAppID iAppId, TelNwkGetLocationType* ioParamP,  
 UINT16* ioTransIdP)`

<b>Parameters</b>	-> <code>iRefnum</code>	The telephony manager library reference number.
	-> <code>iAppId</code>	The telephone application attachment identifier for your application.
	<-> <code>ioParamP</code>	A pointer to a <a href="#">TelNwkGetLocationType</a> structure.

On input, the `size` field of this structure specifies the allocated size of the `value` buffer. Upon return, the `size` field specifies the actual size of the location string, even if it was truncated to fit into the buffer.

`<-> ioTransIdP` Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

`returnCode` `errNone` upon success or an error code upon failure.

`transId` The transaction ID of the operation.

`paramP` Points to the [TelNwkGetLocationType](#) passed to this function in the `ioParamP` parameter.

`functionId` `kTelNwkGetLocationMessage`

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

---

**Comments** The location information string is stored into the `value` field of the [TelNwkGetLocationType](#) structure referenced by `ioParamP`. If the `value` buffer is too small to contain the complete string, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `size` field of the structure is always updated to contain the actual size of the complete string.

Before using this function, you should verify that it is available by calling the [TelIsNwkServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## TelNwkGetNetworkName

**Purpose** Returns the name of a registered network.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelNwkGetNetworkName (UInt16 iRefnum,  
TelAppID iAppId,  
TelNwkGetNetworkNameType* ioParamP,  
UInt16* ioTransIdP)`

**Parameters** `-> iRefnum` The telephony manager library reference number.

`-> iAppId` The telephone application attachment identifier for your application.

`<-> ioParamP` A pointer to a [TelNwkGetNetworkNameType](#) structure that stores the network name.

On input, the `size` field of this structure specifies the allocated size of the value buffer. Upon return, the `size` field specifies the actual size of the name string, even if it was truncated to fit into the buffer.

`<-> ioTransIdP` Set the value of this parameter to `NULL` to cause the function to execute synchronously.

If this parameter is not `NULL`, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the <a href="#">TelNwkGetNetworkNameType</a> passed to this function in the <code>ioParamP</code> parameter.
functionId	<code>kTelNwkGetNetworkNameMessage</code>

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

---

**Comments** The network name string is stored into the `value` field of the structure. If the `value` field buffer in the [TelNwkGetNetworkNameType](#) structure is too small to contain the complete string, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `size` field is always updated to contain the actual size of the complete string.

The string that is returned in the `value` field of the structure referenced by `ioParamP` is network dependent.

On a GSM network, the result string is compliant with the AT 07.07 European Telecommunications Standards Institute (ETSI) standard for COPS and CREG commands. The result string contains the following elements:

- The network type, as returned by the [TelNwkGetNetworkType](#) function, and followed by a semicolon (' ; ') character.
- The network operator, using the following syntax:

<area code> ';' <network registration>

The <area code> value is the value retrieved by issuing the AT+CREG? command.

The <network registration> value is the value retrieved by issuing the AT+CREG? command.

Before using this function, you should verify that it is available by calling the [TelIsNwkServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelNwkGetNetworks](#), [TelNwkGetSelectedNetwork](#)

## TelNwkGetNetworks

**Purpose** Retrieves information about the registered networks.

**Declared In** TelephonyMgr.h

**Prototype** Err TelNwkGetNetworks (UInt16 iRefnum,  
TelAppID iAppId, TelNwkGetNetworksType\* ioParamP,  
UInt16\* ioTransIdP)

**Parameters**

-> iRefnum	The telephony manager library reference number.
-> iAppId	The telephone application attachment identifier for your application.
<-> ioParamP	A pointer to a <a href="#">TelNwkGetNetworksType</a> structure that stores the network IDs. On input, the size field of this structure contains the size, in elements, of the networkIdP array field.  Upon return, the networkIdP array contains the IDs of the registered networks, and the size field contains the number of IDs in the array.

<-> `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

`returnCode` `errNone` upon success or an error code upon failure.

`transId` The transaction ID of the operation.

`paramP` Points to the [TelNwkGetNetworksType](#) passed to this function in the `ioParamP` parameter.

`functionId` `kTelNwkGetNetworkCountMessage`

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

---

**Comments** Before using this function, you should verify that it is available by calling the [TelIsNwkServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelNwkGetNetworkName](#), [TelNwkGetNetworks](#)

## TelNwkGetNetworkType

**Purpose** Retrieve the type of the selected network.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelNwkGetNetworkType (UInt16 iRefnum,  
TelAppID iAppId, UInt8* oTypeP,  
UInt16* ioTransIdP)`

<b>Parameters</b>	-> <code>iRefnum</code>	The telephony manager library reference number.
	-> <code>iAppId</code>	The telephone application attachment identifier for your application.
	<- <code>oTypeP</code>	A pointer to an unsigned byte value. Upon return, this is the network type. This is one of <a href="#">Network Type Constants</a> .
	<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> upon success or an error code upon failure.
<code>transId</code>	The transaction ID of the operation.
<code>paramP</code>	Points to the unsigned integer value passed to this function in the <code>oTypeP</code> parameter.
<code>functionId</code>	<code>kTelNwkGetNetworkTypeMessage</code>

**WARNING!** When using this function asynchronously, you must ensure that the value you pass for the `oTypeP` parameter remains in memory until the asynchronous call completes.

---

- Comments** Before using this function, you should verify that it is available by calling the [TelIsNwkServiceAvailable](#) macro.
- Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## TelNwkGetSearchMode

- Purpose** Returns the current network search mode.
- Declared In** `TelephonyMgr.h`
- Prototype** `Err TelNwkGetSearchMode (UInt16 iRefnum,  
TelAppID iAppId, UInt8* oModeP,  
UInt16* ioTransIdP)`
- Parameters**
- |                             |   |
|-----------------------------|---|
| -> <code>iRefnum</code>     | The telephony manager library reference number.   |
| -> <code>iAppId</code>      | The telephone application attachment identifier for your application.   |
| <- <code>oModeP</code>      | A pointer to an unsigned byte value. Upon return, this is the type of search mode that is currently being used. This is one of the <a href="#">Network Search Mode Constants</a> .  |
| <-> <code>ioTransIdP</code> | Set the value of this parameter to NULL to cause the function to execute synchronously.<br>If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation. |

<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:
returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the unsigned integer value passed to this function in the oModeP parameter.
functionId	kTelNwkGetSearchModeMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by oModeP remains in memory until the asynchronous call completes.

---

<b>Comments</b>	Before using this function, you should verify that it is available by calling the <a href="#">TelIsNwkServiceAvailable</a> macro.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelNwkSetSearchMode</a>

## TelNwkGetSelectedNetwork

<b>Purpose</b>	Retrieve the network identifier of the currently selected network.
<b>Declared In</b>	TelephonyMgr.h
<b>Prototype</b>	Err TelNwkGetSelectedNetwork (UInt16 iRefnum, TelAppID iAppId, UInt32* oNetworkIdP, UInt16* ioTransIdP)
<b>Parameters</b>	-> iRefnum      The telephony manager library reference number.

-> iAppId The telephone application attachment identifier for your application.

<- oNetworkIdP A pointer to an unsigned integer value. Upon return, this is the identifier of the currently selected network.

<-> ioTransIdP Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the unsigned integer value passed to this function in the oNetworkIdP parameter.
functionId	kTelNwkGetSelectedNetworkMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by oNetworkP remains in memory until the asynchronous call completes.

---

**Comments** Before using this function, you should verify that it is available by calling the [TelIsNwkServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelNwkGetNetworkName](#), [TelNwkGetNetworks](#),  
[TelNwkSelectNetwork](#)

## TelNwkGetSignalLevel

**Purpose** Retrieve the selected network carrier signal level.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelNwkGetSignalLevel(UInt16 iRefnum,  
TelAppID iAppId, UInt8* oSignalP,  
UInt16* ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
<- <code>oSignalP</code>	A pointer to an unsigned byte value. Upon return, this is an indication of the signal level in decibels per milliwatt (dBm). The values are explained in the Comments section.
<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the unsigned integer value passed to this function in the oSignalP parameter.
functionId	kTelNwkGetSignalLevelMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by oSignalP remains in memory until the asynchronous call completes.

---

- Comments** This function sets the value of the variable referenced by oSignalP to an integer value that indicates the signal strength in dBm.  
The following table describes the signal strength values.

Signal level value	dBm value
0	$\leq 113$ dBm
1	111 dBm
2 to 30	109 dBm to 53 dBm
31	$\geq 51$ dBm
99	unknown or undetectable

Before using this function, you should verify that it is available by calling the [TelIsNwkServiceAvailable](#) macro.

- Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## TelNwkSelectNetwork

<b>Purpose</b>	Select a network to use from among the set of registered networks.								
<b>Declared In</b>	TelephonyMgr.h								
<b>Prototype</b>	<pre>Err TelNwkSelectNetwork(UInt16 iRefnum,                          TelAppID iAppId, UInt32 iNetworkId,                          UInt16* ioTransIdP)</pre>								
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>-&gt; iNetworkId</td><td>The identifier of the network to select.</td></tr><tr><td>&lt;-&gt; ioTransIdP</td><td>Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</td></tr></table>	-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	-> iNetworkId	The identifier of the network to select.	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
-> iRefnum	The telephony manager library reference number.								
-> iAppId	The telephone application attachment identifier for your application.								
-> iNetworkId	The identifier of the network to select.								
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.								
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.								
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes: <table><tr><td>returnCode</td><td>errNone upon success or an error code upon failure.</td></tr><tr><td>transId</td><td>The transaction ID of the operation.</td></tr><tr><td>paramP</td><td>Points to the network unsigned integer value passed to this function in the iNetworkId parameter.</td></tr><tr><td>functionId</td><td>kTelNwkSelectNetworkMessage</td></tr></table>	returnCode	errNone upon success or an error code upon failure.	transId	The transaction ID of the operation.	paramP	Points to the network unsigned integer value passed to this function in the iNetworkId parameter.	functionId	kTelNwkSelectNetworkMessage
returnCode	errNone upon success or an error code upon failure.								
transId	The transaction ID of the operation.								
paramP	Points to the network unsigned integer value passed to this function in the iNetworkId parameter.								
functionId	kTelNwkSelectNetworkMessage								

<b>Comments</b>	Before using this function, you should verify that it is available by calling the <a href="#">TelIsNwkServiceAvailable</a> macro.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelNwkGetNetworkName</a> , <a href="#">TelNwkGetNetworks</a> , <a href="#">TelNwkGetSelectedNetwork</a>

## TelNwkSetSearchMode

<b>Purpose</b>	Sets the search mode used to find a network.	
<b>Declared In</b>	TelephonyMgr.h	
<b>Prototype</b>	Err TelNwkSetSearchMode (UInt16 iRefnum, TelAppID iAppId, UInt8 iMode, UInt16* ioTransIdP)	
<b>Parameters</b>	-> iRefnum	The telephony manager library reference number.
	-> iAppId	The telephone application attachment identifier for your application.
	-> iMode	The search mode to use. This must be one of the <a href="#">Network Search Mode Constants</a> .
	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.	
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:	

returnCode      errNone upon success or an error code upon failure.

transId      The transaction ID of the operation.

paramP      Points to the unsigned integer value passed to this function in the iMode parameter.

functionId      kTelNwkSetSearchModeMessage

**Comments**      Before using this function, you should verify that it is available by calling the [TelIsNwkServiceAvailable](#) macro.

**Compatibility**      Implemented only if [4.0 New Feature Set](#) is present.

**See Also**      [TelNwkGetSearchMode](#)

## **Telephony Network**

*Telephony Network Functions*

---

# Telephony Calls

---

This chapter describes the telephony calls service sets of the telephony API, which include:

- The data calls service set
- The emergency calls service set
- The speech (voice) calls service set

For more information about the telephony manager basic services and the different service sets, see [Chapter 68, “Telephony Basic Services.”](#)

This chapter describes:

- [Telephony Calls Data Structures](#)
- [Telephony Calls Functions](#)

For more information about using the telephony manager, see [Chapter 10, “Telephony Manager”](#) in *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## Telephony Calls Data Structures

This section describes the data structures used with the telephony calls service sets portion of the telephony manager API.

### TelDtcCallNumberType

The [TelDtcCallNumber](#) function uses the TelDataCallNumberType structure to specify information about the telephone number to call.

```
typedef struct _TelDtcCallNumberType
{
    char      *dialNumberP;
    UInt8     lineId;
} TelDtcCallNumberType
```

## Telephony Calls

### *Telephony Calls Data Structures*

---

#### Field Descriptions

- > dialNumberP      The telephone number to dial.
- <- lineId      Upon return from the [TelDtcCallNumber](#) function, this is the ID of the data line that was established for the telephone call.

## TelDtcReceiveDataType

The [TelDtcReceiveData](#) function uses a TelDtcReceiveDataType structure to receive data from an open data line.

```
typedef struct _TelDtcReceiveDataType
{
    UInt8      *data;
    UInt32     size;
    UInt32     timeout;
} TelDtcReceiveDataType
```

#### Field Descriptions

- <- data      A buffer into which the data is stored.  
  
Note that if this buffer is too small to contain all of the available data, the end of the data is truncated and the function using this structure generates a `telErrBufferSize` error.
- <-> size      When the structure is used as an input parameter, this is the allocated size, in bytes, of the data buffer.  
  
Upon return, this is the actual number of bytes of data that was retrieved. If the data buffer is too small to contain all of the retrieved data, the function using this structure generates a `telErrBufferSize` error.
- > timeout      The number of milliseconds to wait before timing out.

## **TelDtcSendDataType**

The [TelDtcReceiveData](#) function uses a TelDtcSendDataType structure to send data to an open data line.

```
typedef struct _TelDtcSendDataType
{
    UInt8      *data;
    UInt32     size;
} TelDtcSendDataType
```

### **Field Descriptions**

- |    |      |   |
|----|------|---|
| -> | data | A pointer to the data to send.                  |
| -> | size | The number of bytes of data in the data buffer. |

## **TelEmcGetNumberType**

The [TelEmcGetNumber](#) function uses a TelEmcGetNumberType structure to retrieve an emergency dial telephone number.

```
typedef struct _TelEmcGetNumberType
{
    UInt8      index;
    UInt8      size;
    Char       *value;
} TelEmcGetNumberType
```

### **Field Descriptions**

- |    |       |  |
|----|-------|--|
| -> | index | The index of the telephone number. This is a zero-based index. |
|----|-------|--|

## Telephony Calls

### *Telephony Calls Data Structures*

---

<-> size

When the structure is used as an input parameter, this is the allocated size, in bytes, of the value buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the value buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

<- value

A null-terminated string buffer into which the emergency dial telephone number is stored.

Note that if this buffer is too small to contain all of the available data, the end of the data is truncated and the function using this structure generates a `telErrBufferSize` error.

## TelEmcSetNumberType

The [TelEmcSetNumber](#) function uses a `TelEmcNumberType` structure to set an emergency dial telephone number.

```
typedef struct _TelEmcSetNumberType
{
    UInt8    index;
    Char     *value;
} TelEmcSetNumberType
```

### Field Descriptions

-> index

The index of the telephone number. This is a zero-based index.

-> value

The string value of the number to store as the `index`th entry.

## TelSpcGetCallerNumberType

The [TelSpcGetCallerNumber](#) function uses a `TelSpcGetCallerNumberType` structure to retrieve an incoming telephone number.

```
typedef struct _TelSpcGetCallerNumberType
{
    Char     *value;
    UInt8    size;
} TelSpcGetCallerNumberType
```

### Field Descriptions

<- value

A null-terminated string buffer into which the caller telephone number is stored.

Note that if this buffer is too small to contain all of the available data, the end of the data is truncated and the function using this structure generates a `telErrBufferSize` error.

<-> size

When the structure is used as an input parameter, this is the allocated size, in bytes, of the `value` buffer.

Upon return, this is the actual size of the caller dial telephone number, including the null terminator character. If the `value` buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

## TelSpcPlayDTMFType

The [TelSpcPlayDTMF](#) function uses a `TelSpcPlayDTMFType` structure to specify the qualities of the DTMF (dual-tone, multi-frequency) sound sent by the phone to the network or remote, connected equipment.

```
typedef struct _TelSpcPlayDTMFType
{
    UInt8    keyTone;
    UInt32   duration;
} TelSpcPlayDTMFType
```

## Telephony Calls

### Telephony Calls Functions

---

#### Field Descriptions

- > keyTone                         The keycode of the key tone to play. This must be one of the [Keycode Constants](#).
- > duration                         The duration of the tone, specified as a multiple of ten milliseconds.

## Telephony Calls Functions

This section describes the functions used with the telephony calls service sets portion of the telephony API.

### TelDtcCallNumber

<b>Purpose</b>	Initiate a data telephone call.	
<b>Declared In</b>	TelephonyMgr.h	
<b>Prototype</b>	<pre>Err TelDtcCallNumber(UInt16 iRefnum,                      TelAppID iAppId, TelDtcCallNumberType *ioParamP,                      UInt16 *ioTransIdP)</pre>	
<b>Parameters</b>	-> iRefnum	The telephony manager library reference number.
	-> iAppId	The telephone application attachment identifier for your application.
	<-> ioParamP	A pointer to a <a href="#">TelDtcCallNumberType</a> structure that specifies information about the telephone call.
	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode      errNone upon success or an error code upon failure.

transId      The transaction ID of the operation.

paramP      Points to the [TelDtcCallNumberType](#) structure passed to this function in the ioDataCallParamP parameter.

functionId      kTelDtcCallNumberMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioDataCallParamP` remains in memory until the asynchronous call completes.

---

**Comments** Call this function to start a data telephone call.

Before using this function, you should verify that it is available by calling the [TelIsDtcServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelDtcCloseLine](#)

## Telephony Calls

### Telephony Calls Functions

---

## TelDtcCloseLine

**Purpose** Hang up a data line.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelDtcCloseLine(UInt16 iRefnum,  
TelAppID iAppId, UInt8 iLineId,  
UInt16 *ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
-> <code>iLineId</code>	The ID of the line to hang up. This is the ID returned by a previous call to the <a href="#">TelDtcCallNumber</a> function.
<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> upon success or an error code upon failure.
<code>transId</code>	The transaction ID of the operation.
<code>paramP</code>	Points to the unsigned integer value passed to this function in the <code>iLineId</code> parameter.
<code>functionId</code>	<code>kTelDtcCloseLineMessage</code>

<b>Comments</b>	Call this function to end a data telephone call.  Before using this function, you should verify that it is available by calling the <a href="#">TelIsDtcServiceAvailable</a> macro.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelDtcCallNumber</a>

## TelDtcReceiveData

<b>Purpose</b>	Receive data on an opened data communications line.								
<b>Declared In</b>	TelephonyMgr.h								
<b>Prototype</b>	Err TelDtcReceiveData(UINT16 iRefnum, TelAppID iAppId, TelDtcReceiveDataType *ioParamP, UINT16 *ioTransIdP)								
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>&lt;-&gt; ioParamP</td><td>A pointer to a <a href="#">TelDtcReceiveDataType</a> structure.</td></tr><tr><td>&lt;-&gt; ioTransIdP</td><td>Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</td></tr></table>	-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	<-> ioParamP	A pointer to a <a href="#">TelDtcReceiveDataType</a> structure.	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
-> iRefnum	The telephony manager library reference number.								
-> iAppId	The telephone application attachment identifier for your application.								
<-> ioParamP	A pointer to a <a href="#">TelDtcReceiveDataType</a> structure.								
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.								
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.								
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:								

## Telephony Calls

### Telephony Calls Functions

---

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the <a href="#">TelDtcReceiveData</a> structure passed to this function in the ioRcvDataP parameter.
functionId	kTelDtcReceiveDataMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

---

**Comments** Call this function to receive data during an active data telephone call.

Before using this function, you should verify that it is available by calling the [TelIsDtcServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelDtcCallNumber](#), [TelDtcCloseLine](#), [TelDtcSendData](#)

## TelDtcSendData

**Purpose** Send data on an opened data line.

**Declared In** TelephonyMgr.h

**Prototype** Err TelDtcSendData(UInt16 iRefnum,  
TelAppID iAppId, TelDtcSendData \*iParamP,  
UInt16 \*ioTransIdP)

**Parameters** -> iRefnum The telephony manager library reference number.

-> iAppId	The telephone application attachment identifier for your application.
-> iParamP	A pointer to a <a href="#">TelDtcSendDataType</a> structure that specifies the data to send.
<-> ioTransIDP	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the <a href="#">TelDtcSendDataType</a> structure passed to this function in the iParamP parameter.
functionId	kTelDtcSendDataMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by iParamP remains in memory until the asynchronous call completes.

---

**Comments** Call this function to send data during an active data telephone call. Before using this function, you should verify that it is available by calling the [TelIsDtcServiceAvailable](#) macro.

## Telephony Calls

### Telephony Calls Functions

---

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelDtcCallNumber](#), [TelDtcCloseLine](#), [TelDtcReceiveData](#)

## TelEmcCall

**Purpose** Call the currently selected emergency service.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelEmcCall (UInt16 iRefnum, TelAppID iAppId,  
UInt16 *ioTransIdP)`

**Parameters**

<code>-&gt; iRefnum</code>	The telephony manager library reference number.
<code>-&gt; iAppId</code>	The telephone application attachment identifier for your application.
<code>&lt;-&gt; ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> upon success or an error code upon failure.
<code>transId</code>	The transaction ID of the operation.
<code>paramP</code>	A NULL pointer.
<code>functionId</code>	<code>kTelEmcCallMessage</code>

**Comments** This function calls the telephone number specified in a previous call to the [TelEmcSelectNumber](#) function. In synchronous mode, this function returns after the dial command has been sent to the phone. After calling this function, sub-launched applications can receive notifications when the following telephony events occur. Note that these notifications can be raised after both synchronous and asynchronous calls to this function.

<b>Event</b>	<b>Description</b>
sysTelSpcLaunchCmdCallReleased	Warns that the telephone call has been released.
sysTelSpcLaunchCmdCallBusy	Warns that the called equipment is busy.
sysTelSpcLaunchCmdCallConnect	Warns that the line is open. The ID of the open line is stored in the <code>UInt32</code> value of the parameter block passed to the application.
sysTelSpcLaunchCmdCallError	Warns that an error occurred while attempting to complete the telephone call.

Before using this function, you should verify that it is available by calling the [TelIsEmcServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelEmcCloseLine](#),

## Telephony Calls

### *Telephony Calls Functions*

---

## TelEmcCloseLine

**Purpose** Close the line that is currently opened for an emergency telephone call.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelEmcCloseLine(UInt16 iRefnum,  
TelAppID iAppId, UInt16 *ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> upon success or an error code upon failure.
<code>transId</code>	The transaction ID of the operation.
<code>paramP</code>	A NULL pointer.
<code>functionId</code>	<code>kTelDtcCloseLineMessage</code>

**Comments** Call this function to end an emergency telephone call.

Before using this function, you should verify that it is available by calling the [TelIsEmcServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelEmcCall](#)

## TelEmcGetNumber

**Purpose** Retrieve an emergency dial telephone number.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelEmcGetNumber(UInt16 iRefnum,  
TelAppID iAppId, TelEmcGetNumberType *ioParamP,  
UInt16 *ioTransIdP)`

<b>Parameters</b>	<p>-&gt; <code>iRefnum</code> The telephony manager library reference number.</p> <p>-&gt; <code>iAppId</code> The telephone application attachment identifier for your application.</p> <p>&lt;-&gt; <code>ioParamP</code> A pointer to a <a href="#">TelEmcGetNumberType</a> structure in which you assign the index of the telephone number that you want to retrieve. On input, the <code>size</code> field of this structure specifies the allocated size of the value buffer. Upon return, the <code>size</code> field specifies the actual size of the telephone number, even if it was truncated to fit into the buffer.</p> <p>&lt;-&gt; <code>ioTransIdP</code> Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</p>
<b>Synchronous Result</b>	Returns <code>errNone</code> if the function was successful or returns an error code if not successful.

## Telephony Calls

### Telephony Calls Functions

---

#### Asynchronous Result

The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the <a href="#">TelEmcGetNumberType</a> structure passed to this function in the <code>ioGetNumberP</code> parameter.
functionId	<code>kTelEmcGetNumberMessage</code>

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

---

#### Comments

The emergency call telephone number is stored into the `value` field of the [TelEmcGetNumberType](#) structure referenced by `ioGetNumberP`. If the `value` buffer is too small to contain the complete string, the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `size` field of the structure is always updated to contain the actual size of the complete string. Before using this function, you should verify that it is available by calling the [TelIsEmcServiceAvailable](#) macro.

#### Compatibility

Implemented only if [4.0 New Feature Set](#) is present.

#### See Also

[TelEmcGetNumberCount](#), [TelEmcSetNumber](#),  
[TelEmcSelectNumber](#)

## TelEmcGetNumberCount

<b>Purpose</b>	Retrieve the count of emergency dial telephone numbers.	
<b>Declared In</b>	TelephonyMgr.h	
<b>Prototype</b>	<pre>Err TelEmcGetNumberCount (UInt16 iRefnum,                            TelAppID iAppId, UInt8 *oCountP,                            UInt16 *ioTransIdP)</pre>	
<b>Parameters</b>	-> iRefnum	The telephony manager library reference number.
	-> iAppId	The telephone application attachment identifier for your application.
	<- oCountP	Upon return, the total number of emergency call numbers available.
	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.	
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:	
returnCode	errNone upon success or an error code upon failure.	
transId	The transaction ID of the operation.	
paramP	Points to unsigned integer passed to this function in the oCountP parameter.	
functionId	kTelEmcGetNumberMessage	

## Telephony Calls

### Telephony Calls Functions

---

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by `oCountP` remains in memory until the asynchronous call completes.

---

<b>Comments</b>	The emergency telephone call number is stored into the <code>value</code> field of the <a href="#">TelEmcGetNumberType</a> structure referenced by <code>ioGetNumberP</code> . If the <code>value</code> buffer is too small to contain the complete string, the string is truncated (and ends with the null terminator character) and this function returns the <code>telErrBufferSize</code> error. The <code>size</code> field of the structure is always updated to contain the actual size of the complete string.  Before using this function, you should verify that it is available by calling the <a href="#">TelIsEmcServiceAvailable</a> macro.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelEmcGetNumber</a> , <a href="#">TelEmcSetNumber</a> , <a href="#">TelEmcSelectNumber</a>

## TelEmcSelectNumber

<b>Purpose</b>	Select the current emergency telephone number. This is the telephone number that gets dialed when you call the <a href="#">TelEmcCall</a> function.						
<b>Declared In</b>	<code>TelephonyMgr.h</code>						
<b>Prototype</b>	<pre>Err TelEmcSelectNumber (UInt16 iRefnum,                       TelAppID iAppId, UInt8 iIndex,                       UInt16 *ioTransIdP)</pre>						
<b>Parameters</b>	<table><tr><td>-&gt; <code>iRefnum</code></td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; <code>iAppId</code></td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>-&gt; <code>iIndex</code></td><td>The zero-based index of the emergency telephone number that you want selected.</td></tr></table>	-> <code>iRefnum</code>	The telephony manager library reference number.	-> <code>iAppId</code>	The telephone application attachment identifier for your application.	-> <code>iIndex</code>	The zero-based index of the emergency telephone number that you want selected.
-> <code>iRefnum</code>	The telephony manager library reference number.						
-> <code>iAppId</code>	The telephone application attachment identifier for your application.						
-> <code>iIndex</code>	The zero-based index of the emergency telephone number that you want selected.						

<-> `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

`returnCode`      `errNone` upon success or an error code upon failure.

`transId`      The transaction ID of the operation.

`paramP`      Points to the unsigned integer value passed to this function in the `iIndex` parameter.

`functionId`      `kTelEmcSelectNumberMessage`

**Comments** Before using this function, you should verify that it is available by calling the [TelIsEmcServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelEmcCall](#), [TelEmcGetNumber](#), [TelEmcGetNumberCount](#), [TelEmcSetNumber](#)

## Telephony Calls

### Telephony Calls Functions

---

## TelEmcSetNumber

**Purpose** Set the telephone number for the specified emergency dial number.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelEmcSetNumber(UInt16 iRefnum,  
TelAppID iAppId, TelEmcSetNumberType *iParamP,  
UInt16 *ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
-> <code>iParamP</code>	A pointer to a <a href="#">TelEmcSetNumberType</a> structure that specifies the telephone number.
<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> upon success or an error code upon failure.
<code>transId</code>	The transaction ID of the operation.
<code>paramP</code>	Points to the <a href="#">TelEmcSetNumberType</a> structure passed to this function in the <code>iNumberP</code> parameter.
<code>functionId</code>	<code>kTelEmcSetNumberMessage</code>

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `iParamP` remains in memory until the asynchronous call completes.

---

**Comments** Call this function to associate a new telephone number with the emergency dial number that has the specified `iIndex`.  
Before using this function, you should verify that it is available by calling the [TelIsEmcServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelEmcGetNumber](#),

## TelSpcAcceptCall

**Purpose** Accept an incoming voice telephone call.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelSpcAcceptCall(UInt16 iRefnum,  
TelAppID iAppId, UInt8 *oLineIdP,  
UInt16 *ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
<- <code>oLineIdP</code>	A pointer to an unsigned byte value. Upon return, this is the ID of the voice line assigned to the telephone call.
<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.

## Telephony Calls

### Telephony Calls Functions

---

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode      errNone upon success or an error code upon failure.

transId      The transaction ID of the operation.

paramP      Points to the unsigned integer passed to this function in the oLineIdP parameter.

functionId      kTelSpcAcceptCallMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by oLineIdP remains in memory until the asynchronous call completes.

---

**Comments** If another line was active prior to the execution of this function, that line is put on hold. Note that there can only be one line active at any given time, and there can only be one line on hold at any given time.

Before using this function, you should verify that it is available by calling the [TelIsSpcServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSpcGetCallerNumber](#), [TelSpcRejectCall](#)

## TelSpcCallNumber

**Purpose** Initiate a voice telephone call.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelSpcCallNumber(UInt16 iRefnum,  
TelAppID iAppId, const Char *iDialNumberP,  
UInt16 *ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
-> <code>iDialNumberP</code>	A pointer to the telephone number to call.
<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> upon success or an error code upon failure.
<code>transId</code>	The transaction ID of the operation.
<code>paramP</code>	Points to the string passed to this function in the <code>iDialNumberP</code> parameter.
<code>functionId</code>	<code>kTelSpcCallNumberMessage</code>

## Telephony Calls

### *Telephony Calls Functions*

---

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by `iDialNumberP` remains in memory until the asynchronous call completes.

---

#### Comments

A successful return from a synchronous call or receipt of a successful notification from an asynchronous call does not mean that the telephone call has been connected; instead, it indicates that the dial command was sent to the phone. Successful connection of the telephone call is indicated with a sub-launch.

The dial number is formatted according to the following syntax:

```
DialNumber ::= <Phone_Number> | <Code_String>
             | <Phone_Number> <Code_String>
```

```
Phone_Number ::= <IntlPrefix><NatlNumber>
                | <NatlNumber>
```

```
IntlPrefix ::= '+' <country code>
```

```
NatlNumber ::= { { Pause<Pause> } { <Digit> } }
```

```
Code_String ::= <Symbol>{ <Symbol> }
```

```
Symbol ::= <Digit> | '#' | '*'
```

```
Digit ::= '0' | '1' | '2' | '3' | '4'
          | '5' | '6' | '7' | '8' | '9'
```

```
Pause ::= ','
```

After calling this function, sub-launched applications can receive notifications when the following telephony events occur. Note that these notifications can be raised after both synchronous and asynchronous calls to this function.

Event	Description
sysTelSpcLaunchCmdCallReleased	This is passed to a sub-launched application to warn that the telephone call has been released.
sysTelSpcLaunchCmdCallBusy	This is passed to a sub-launched application to warn that the called equipment is busy.
sysTelSpcLaunchCmdCallConnect	This is passed to a sub-launched application to warn that the line is open. The ID of the open line is stored in the UInt32 value of the parameter block passed to the application.
sysTelSpcLaunchCmdCallError	This is passed to a sub-launched application to warn that the telephone call encountered an error.

Before using this function, you should verify that it is available by calling the [TelIsSpcServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSpcCloseLine](#)

## TelSpcCloseLine

**Purpose** Ends a voice telephone call.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelSpcCloseLine(UInt16 iRefnum,  
 TelAppID iAppId, UInt8 iLineId,  
 UInt16 *ioTransIdP)`

**Parameters** `-> iRefnum` The telephony manager library reference number.

## Telephony Calls

### Telephony Calls Functions

---

-> iAppId	The telephone application attachment identifier for your application.
-> iLineId	The ID of the voice line that you want to close. This is the ID returned by a previous call to the <a href="#">TelSpcAcceptCall</a> function.
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the unsigned integer value passed to this function in the iLineId parameter.
functionId	kTelSpcCloseLineMessage

**Comments** Before using this function, you should verify that it is available by calling the [TelIsSpcServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSpcCallNumber](#)

## TelSpcConference

<b>Purpose</b>	Initiate a conference telephone call by merging the active line and the held line.	
<b>Declared In</b>	TelephonyMgr.h	
<b>Prototype</b>	<pre>Err TelSpcConference(UInt16 iRefnum,                      TelAppID iAppId, UInt8 *oLineIdP,                      UInt16 *ioTransIdP)</pre>	
<b>Parameters</b>	-> iRefnum	The telephony manager library reference number.
	-> iAppId	The telephone application attachment identifier for your application.
	<- oLineIdP	A pointer to an unsigned byte value. Upon return, this is the ID of the voice line assigned to the telephone call.
	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.	
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:  returnCode      errNone upon success or an error code upon failure.  transId      The transaction ID of the operation.	

## Telephony Calls

### Telephony Calls Functions

---

paramP	Points to the unsigned integer value passed to this function in the <code>oLineIdP</code> parameter.
functionId	<code>kTelSpcConferenceMessage</code>

---

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by `oLineIdP` remains in memory until the asynchronous call completes.

---

<b>Comments</b>	Before using this function, you should verify that it is available by calling the <a href="#">TelIsSpcServiceAvailable</a> macro.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelSpcCallNumber</a> , <a href="#">TelSpcCloseLine</a> , <a href="#">TelSpcHoldLine</a> , <a href="#">TelSpcRetrieveHeldLine</a> , <a href="#">TelSpcSelectLine</a>

## TelSpcGetCallerNumber

<b>Purpose</b>	Retrieve the telephone number of the caller on an incoming telephone call.				
<b>Declared In</b>	<code>TelephonyMgr.h</code>				
<b>Prototype</b>	<pre>Err TelSpcGetCallerNumber(UInt16 iRefnum,                            TelAppID iAppId,                            TelSpcGetCallerNumberType *ioParamP,                            UInt16 *ioTransIdP)</pre>				
<b>Parameters</b>	<table><tr><td>-&gt; <code>iRefnum</code></td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; <code>iAppId</code></td><td>The telephone application attachment identifier for your application.</td></tr></table>	-> <code>iRefnum</code>	The telephony manager library reference number.	-> <code>iAppId</code>	The telephone application attachment identifier for your application.
-> <code>iRefnum</code>	The telephony manager library reference number.				
-> <code>iAppId</code>	The telephone application attachment identifier for your application.				

<-> ioParamP	A pointer to a <a href="#">TelSpcGetCallerNumberType</a> structure that is used to retrieve the caller's telephone number.  On input, the size field of this structure specifies the allocated size of the value buffer. Upon return, the size field specifies the actual size of the telephone number, even if it was truncated to fit into the buffer.
<-> ioTransIDP	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the function was successful or returns an error code if not successful. If there is no active incoming telephone call, this function returns the telErrUnavailableValue error.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the <a href="#">TelSpcGetCallerNumberType</a> structure passed to this function in the ioParamP parameter.
functionId	kTelSpcGetCallerNumberMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by ioParamP remains in memory until the asynchronous call completes.

---

## Telephony Calls

### Telephony Calls Functions

---

<b>Comments</b>	The emergency telephone call number is stored into the value field of the <a href="#">TelSpcGetCallerNumberType</a> structure referenced by ioParamP. If the value buffer is too small to contain the complete string, the string is truncated (and ends with the null terminator character) and this function returns the telErrBufferSize error. The size field of the structure is always updated to contain the actual size of the complete string.  Before using this function, you should verify that it is available by calling the <a href="#">TelIsSpcServiceAvailable</a> macro.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelSpcAcceptCall</a>

## TelSpcHoldLine

<b>Purpose</b>	Put the currently active voice line on hold.	
<b>Declared In</b>	TelephonyMgr.h	
<b>Prototype</b>	Err TelSpcHoldLine(UInt16 iRefnum, TelAppID iAppId, UInt16 *ioTransIdP)	
<b>Parameters</b>	-> iRefnum	The telephony manager library reference number.
	-> iAppId	The telephone application attachment identifier for your application.
	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.	

<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:
returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	A NULL pointer.
functionId	kTelSpcHoldLineMessage
<b>Comments</b>	Note that there can only be one line active at any given time, and there can only be one line on hold at any given time.  Before using this function, you should verify that it is available by calling the <a href="#">TelIsSpcServiceAvailable</a> macro.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelSpcRetrieveHeldLine</a>

## TelSpcPlayDTMF

<b>Purpose</b>	Play a dual-tone multi-frequency sound to the network system for a specified duration. Note that you can only play a DTMF while a voice telephone call is active.				
<b>Declared In</b>	TelephonyMgr.h				
<b>Prototype</b>	<pre>Err TelSpcPlayDTMF(UINT16 iRefnum,                      TelAppID iAppId, TelSpcPlayDTMFType *iParamP,                      UInt16 *ioTransIdP)</pre>				
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr></table>	-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.
-> iRefnum	The telephony manager library reference number.				
-> iAppId	The telephone application attachment identifier for your application.				

## Telephony Calls

### Telephony Calls Functions

---

-> iParamP	A pointer to a <a href="#">TelSpcPlayDTMFType</a> structure that specifies the tone to play and its duration.
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the <a href="#">TelSpcPlayDTMFType</a> structure passed to this function in the iParamP parameter
functionId	kTelSpcPlayDTMFMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by iParamP remains in memory until the asynchronous call completes.

---

**Comments** Before using this function, you should verify that it is available by calling the [TelIsSpcServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSpcSendBurstDTMF](#), [TelSpcStartContinuousDTMF](#), [TelSpcStopContinuousDTMF](#)

## **TelSpcRejectCall**

<b>Purpose</b>	Reject an incoming voice telephone call.								
<b>Declared In</b>	TelephonyMgr.h								
<b>Prototype</b>	<pre>Err TelSpcRejectCall(UInt16 iRefnum, TelAppID iAppId, UInt16 *ioTransIdP)</pre>								
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>&lt;-&gt; ioTransIdP</td><td>Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</td></tr></table>	-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.		
-> iRefnum	The telephony manager library reference number.								
-> iAppId	The telephone application attachment identifier for your application.								
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.								
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.								
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes: <table><tr><td>returnCode</td><td>errNone upon success or an error code upon failure.</td></tr><tr><td>transId</td><td>The transaction ID of the operation.</td></tr><tr><td>paramP</td><td>A NULL pointer.</td></tr><tr><td>functionId</td><td>kTelSpcRejectCallMessage</td></tr></table>	returnCode	errNone upon success or an error code upon failure.	transId	The transaction ID of the operation.	paramP	A NULL pointer.	functionId	kTelSpcRejectCallMessage
returnCode	errNone upon success or an error code upon failure.								
transId	The transaction ID of the operation.								
paramP	A NULL pointer.								
functionId	kTelSpcRejectCallMessage								
<b>Comments</b>	Before using this function, you should verify that it is available by calling the <a href="#">TelIsSpcServiceAvailable</a> macro.								

## Telephony Calls

### Telephony Calls Functions

---

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSpcAcceptCall](#), [TelSpcGetCallerNumber](#)

## TelSpcRetrieveHeldLine

**Purpose** Reconnect the voice line that is currently on hold, making it the active voice line.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelSpcRetrieveHeldLine(UInt16 iRefnum,  
TelAppID iAppId, UInt16 *ioTransIdP)`

**Parameters**

<code>-&gt; iRefnum</code>	The telephony manager library reference number.
<code>-&gt; iAppId</code>	The telephone application attachment identifier for your application.
<code>&lt;-&gt; ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> upon success or an error code upon failure.
<code>transId</code>	The transaction ID of the operation.
<code>paramP</code>	A NULL pointer.
<code>functionId</code>	<code>kTelSpcRetrieveHeldLineMessage</code>

<b>Comments</b>	Note that there can only be one line active at any given time, and there can only be one line on hold at any given time.  Before using this function, you should verify that it is available by calling the <a href="#">TelIsSpcServiceAvailable</a> macro.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelSpcHoldLine</a>

## TelSpcSelectLine

<b>Purpose</b>	Select the specified line ID as the newly active voice line.								
<b>Declared In</b>	TelephonyMgr.h								
<b>Prototype</b>	Err TelSpcSelectLine(UInt16 iRefnum, TelAppID iAppId, UInt8 iLineId, UInt16 *ioTransIdP)								
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>-&gt; iLineId</td><td>The ID of the voice line that you want to activate. This is the ID returned by a previous call to the <a href="#">TelSpcAcceptCall</a> function.</td></tr><tr><td>&lt;-&gt; ioTransIdP</td><td>Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</td></tr></table>	-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	-> iLineId	The ID of the voice line that you want to activate. This is the ID returned by a previous call to the <a href="#">TelSpcAcceptCall</a> function.	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
-> iRefnum	The telephony manager library reference number.								
-> iAppId	The telephone application attachment identifier for your application.								
-> iLineId	The ID of the voice line that you want to activate. This is the ID returned by a previous call to the <a href="#">TelSpcAcceptCall</a> function.								
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.								
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.								

## Telephony Calls

### Telephony Calls Functions

---

<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:
returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the unsigned integer value passed to this function in the <code>iLineId</code> parameter.
functionId	<code>kTelSpcSelectLineMessage</code>
<b>Comments</b>	If a line was active previous to completion of this function, that line is put on hold. Note that there can only be one line active at any given time, and there can only be one line on hold at any given time. Before using this function, you should verify that it is available by calling the <a href="#">TelIsSpcServiceAvailable</a> macro.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelSpcConference</a>

## TelSpcSendBurstDTMF

<b>Purpose</b>	Send a string of dual-tone, multi-frequency sounds to the network system. Note that you can only play a DTMF while a voice telephone call is active.				
<b>Declared In</b>	<code>TelephonyMgr.h</code>				
<b>Prototype</b>	<pre>Err TelSpcSendBurstDTMF (UInt16 iRefnum,                            TelAppID iAppId, const Char *iDTMFStringP,                            UInt16 *ioTransIdP)</pre>				
<b>Parameters</b>	<table><tr><td>-&gt; <code>iRefnum</code></td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; <code>iAppId</code></td><td>The telephone application attachment identifier for your application.</td></tr></table>	-> <code>iRefnum</code>	The telephony manager library reference number.	-> <code>iAppId</code>	The telephone application attachment identifier for your application.
-> <code>iRefnum</code>	The telephony manager library reference number.				
-> <code>iAppId</code>	The telephone application attachment identifier for your application.				

-> `iDTMFStringP`  
A null-terminated string of keytone values.  
Each byte of the string specifies one of the  
[Keycode Constants](#).

<-> `ioTransIDP` Set the value of this parameter to NULL to cause the function to execute synchronously.  
If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

`returnCode`      `errNone` upon success or an error code upon failure.

`transId`      The transaction ID of the operation.

`paramP`      Points to the string passed to this function in the `iDTMFStringP` parameter.

`functionId`      `kTelSpcSendBurstDTMFMessage`

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `iDTMFStringP` remains in memory until the asynchronous call completes.

---

**Comments** This function sends a burst string of keytones to the network. Each key tone is played for the default duration defined by the network. Before using this function, you should verify that it is available by calling the [TelIsSpcServiceAvailable](#) macro.

## Telephony Calls

### Telephony Calls Functions

---

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSpcPlayDTMF](#), [TelSpcStartContinuousDTMF](#),  
[TelSpcStopContinuousDTMF](#)

## TelSpcStartContinuousDTMF

**Purpose** Send a continuous dual-tone, multi-frequency sound to the network system. Note that you can only play a DTMF while a voice telephone call is active.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelSpcStartContinuousDTMF(UInt16 iRefnum,  
TelAppID iAppId, UInt8 iKeyCode,  
UInt16 *ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
-> <code>iKeyCode</code>	The keycode to send to the network. This must be one of the <a href="#">Keycode Constants</a> .
<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the unsigned integer value passed to this function in the iKeyCode parameter.
functionId	kTelSpcStartContinuousDTMFMessage

**Comments** This function sends a key tone to the network system that is played continuously until the [TelSpcStopContinuousDTMF](#) function executes.  
Before using this function, you should verify that it is available by calling the [TelIsSpcServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSpcPlayDTMF](#), [TelSpcSendBurstDTMF](#),  
[TelSpcStopContinuousDTMF](#)

## TelSpcStopContinuousDTMF

**Purpose** Stop the continuous playing of a tone that was started by calling the [TelSpcStartContinuousDTMF](#) function.

**Declared In** TelephonyMgr.h

**Prototype** Err TelSpcStopContinuousDTMF (UInt16 iRefnum,  
TelAppID iAppId, UInt16 \*ioTransIdP)

**Parameters**

-> iRefnum	The telephony manager library reference number.
-> iAppId	The telephone application attachment identifier for your application.
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously.

## Telephony Calls

### Telephony Calls Functions

---

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode      errNone upon success or an error code upon failure.

transId      The transaction ID of the operation.

paramP      A NULL pointer.

functionId      kTelSpcStopContinuousDTMFMessage

**Comments** This function stops the continuous playing of the tone that was previously initiated by calling the [TelSpcStartContinuousDTMF](#) function.

Before using this function, you should verify that it is available by calling the [TelIsSpcServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSpcPlayDTMF](#), [TelSpcSendBurstDTMF](#),  
[TelSpcStartContinuousDTMF](#)

# Telephony SMS

---

This chapter describes the telephony SMS service set of the telephony API.

For more information about the telephony manager basic services and the different service sets, see [Chapter 68, “Telephony Basic Services.”](#)

This chapter describes:

- [Telephony SMS Data Structures](#)
- [Telephony SMS Constants](#)
- [Telephony SMS Functions](#)

For more information about using the telephony manager, see [Chapter 10, “Telephony Manager”](#) in *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## Telephony SMS Data Structures

This section describes the data structures used with the SMS service set of the telephony API.

### TelSmsDateTimeType

Several of the other data structures used with the Telephony SMS functions include an `TelSmsDateTimeType` structure to store a date and time value.

```
typedef _TelSmsDateTimeType
    Boolean    absolute;
    UInt32    dateTIme;
} TelSmsDateTimeType
```

## Telephony SMS

### Telephony SMS Data Structures

---

#### Field Descriptions

- > absolute If true, the dateTime value is a Palm™ absolute time value, which is the number of seconds since 1/1/1904. If false, the dateTime value is relative to the current date and time.
- > dateTime The date and time value.
  - If the absolute field is true, this is expressed as the number of seconds elapsed since 12:00 A.M. on January 1, 1904. This is the format returned by the [TimGetSeconds](#) function.
  - If the absolute field is false, this is expressed as the number of seconds elapsed from the current time.

## TelSmsDeleteMessageType

The [TelSmsDeleteMessage](#) function uses a TelSmsDeleteMessageType structure to specify the message to be deleted.

```
typedef struct _TelSmsDeleteMessageType
{
    UInt8    messageType;
    UInt16   index;
} TelSmsDeleteMessageType
```

#### Field Descriptions

- > messageType The message type. This is one of the [SMS Message Type Constants](#).
  - > index The index of the SMS message in the phone's storage that is to be deleted. Note that the message is deleted from the storage area selected by a call to the [TelSmsSelectStorage](#) function.
- Note that the index is zero-based.

## **TelSmsDeliveryAdvancedCDMAType**

The [TelSmsDeliveryMessageType](#) structure includes a **TelSmsDeliveryAdvancedCDMAType** structure for CDMA messages.

```
typedef struct _TelSmsDeliveryAdvancedCDMAType
{
    UInt8          messageType;
    TelSmsDateTimeType validityPeriod;
    UInt8          priority;
    UInt8          privacy;
    Boolean        alertOnDeliveryRequest;
    Boolean        manualAckRequest;
    UInt8          voiceMessageNumber;
    UInt8          callbackNumberSize;
    Char           *callbackNumberAddress;
    UInt8          languageIndicator;
} TelSmsDeliveryAdvancedCDMAType
```

### **Field Descriptions**

messageType	The type of the message. This must be one of the <a href="#">SMS Message Type Constants</a> .
validityPeriod	An <a href="#">TelSmsDateTimeType</a> structure that specifies the amount of time for which the message is valid.
priority	The message priority. This must be one of the <a href="#">SMS Message Urgency Constants</a> .
privacy	The privacy type of the message. This must be one of the <a href="#">SMS Message Privacy Constants</a> .
alertOnDeliveryRequest	true if the user is to be alerted upon delivery of this message, and false if not.
manualAckRequest	true if a reply is requested from the user, and false if not.
voiceMessageNumber	

callbackNumberSize

When the structure is used as an input parameter, this is the allocated size, in bytes, of the callbackNumberAddress string.

Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a telErrBufferSize error.

callbackNumberAddress

A buffer into which the callback number address string is stored.

Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a telErrBufferSize error.

languageIndicator

## TelSmsDeliveryAdvancedGSMTypE

The [TelSmsDeliveryMessageType](#) structure includes a TelSmsDeliveryAdvancedGSMTypE structure for GSM messages.

```
typedef struct _TelSmsDeliveryAdvancedGSMTypE
{
    UInt16      protocolId;
    Boolean     replyPath;
    Char        *serviceCenterNumber;
    UInt8       serviceCenterNumberSize;
} TelSmsDeliveryAdvancedGSMTypE
```

### Field Descriptions

<- protocolId	The protocol used for this message. This is one of the <a href="#">SMS Message Transport Protocol Constants</a> .
<- replyPath	If this value is set, then you use the serviceCenterNumber to reply to this message.
	If this value is not set, you use the default service center provided by your network operator.
<-> serviceCenterNumber	A buffer into which the service center number string is stored.
	Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a telErrBufferSize error.
<-> serviceCenterNumberSize	When the structure is used as an input parameter, this is the allocated size, in bytes, of the serviceCenterNumber string.  Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a telErrBufferSize error.

## TelSmsDeliveryAdvancedTDMAType

The [TelSmsDeliveryMessageType](#) structure includes a TelSmsDeliveryAdvancedTDMAType structure for TDMA messages.

```
typedef struct _TelSmsDeliveryAdvancedTDMAType
{
    UInt8           messageType;
    TelSmsDateTimeType validityPeriod;
    UInt8           priority;
```

## Telephony SMS

### Telephony SMS Data Structures

---

```
    UInt8           privacy;
    Boolean        alertOnDeliveryRequest;
    Boolean        manualAckRequest;
    UInt8          voiceMessageNumber;
    UInt8          callbackNumberSize;
    Char           *callbackNumberAddress;
    UInt8          languageIndicator;
} TelSmsDeliveryAdvancedTDMAType
```

#### Field Descriptions

messageType	The type of the message. This must be one of the <a href="#">SMS Message Type Constants</a> .
validityPeriod	An <a href="#">TelSmsDateTimeType</a> structure that specifies the amount of time for which the message is valid.
priority	The message priority. This must be one of the <a href="#">SMS Message Urgency Constants</a> .
privacy	The privacy type of the message. This must be one of the <a href="#">SMS Message Privacy Constants</a> .
alertOnDeliveryRequest	true if the user is to be alerted upon delivery of this message, and false if not.
manualAckRequest	true if a reply is requested from the user, and false if not.
voiceMessageNumber	
callbackNumberSize	When the structure is used as an input parameter, this is the allocated size, in bytes, of the callbackNumberAddress string.  Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a telErrBufferSize error.

callbackNumberAddress A buffer into which the callback telephone number address string is stored.

Note that if this buffer is too small to contain the entire retrieved string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a `telErrBufferSize` error.

languageIndicator

## TelSmsDeliveryMessageType

The [TelSmsReadMessage](#) function uses a `TelSmsDeliveryMessageType` structure to retrieve information about a delivered message.

```
typedef struct _TelSmsDeliveryMessageType
{
    UInt16           version;
    UInt16           index;
    UInt32           messageIdentifier;
    TelSmsDateTimeType timeStamp;
    UInt16           dataSize;
    UInt8            *data;
    UInt8           dataCodingScheme;
    UInt8           originatingAddressSize;
    Char             *originatingAddress;
    Boolean          otherToReceive;
    Boolean          reportDeliveryIndicator;
    UInt8           standardType;

    union
    {
        TelSmsDeliveryAdvancedGSMTyp
                                advancedGSM;
        TelSmsDeliveryAdvancedCDMATyp
                                advancedCDMA;
        TelSmsDeliveryAdvancedTDMATyp
                                advancedTDMA;
    } advancedParams;
```

## Telephony SMS

### Telephony SMS Data Structures

---

```
    UInt8           extensionsCount;
    TelSmsExtensionType *extensionsP;
} TelSmsDeliveryMessageType
```

#### Field Descriptions

-> version	The version of the SMS API associated with this message.
<-> index	Upon return, the SMS index of the message on the phone. This is a 0-based index.
	This value is used for input only when calling the <a href="#">TelSmsReadMessage</a> function to read one SMS at a time.
<- messageIdentifier	The message identifier.
<- timeStamp	The message time stamp. This is a <a href="#">TelSmsDateTimeType</a> structure.
<-> dataSize	The size of the data buffer.
	When the structure is used as an input parameter, this is the allocated size of the data buffer.
	Upon return, this is the actual size of the message data. If the buffer is too small to contain the entire message, this field is assigned the entire length of the message, and the function using this structure generates a <code>telErrBufferSize</code> error.
<-> data	A buffer into which the retrieved data is stored.
	Note that if this buffer is too small to contain the entire retrieved message, the end of the message is truncated and the function using this structure generates a <code>telErrBufferSize</code> error.
<- dataCodingScheme	The coding scheme used for the data. This is one of the <a href="#">SMS Data Coding Scheme Constants</a> .

<-> originatingAddressSize The size of the originatingAddress buffer.

When the structure is used as an input parameter, this is the allocated size of the originatingAddress buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a telErrBufferSize error.

<-> originatingAddress

A buffer into which the originating address string is stored.

Note that if this buffer is too small to contain the entire string, the end of the string is truncated (and ends with the null terminator character) and the function using this structure generates a telErrBufferSize error.

<- otherToReceive

Indicates whether there are more messages waiting to be received from the service center to the mobile device.

<- reportDeliveryIndicator

If true, indicates that the originating user has asked the network to send a delivery report.

<- standardType

Indicates which field of the advancedParams union contains the message information. This is one of the [Network Type Constants](#) described in [Chapter 70, “Telephony Network.”](#)

If this value is kTelNwkCDMA, then the advancedParams union contains a [TelSmsDeliveryAdvancedCDMAType](#) structure, and similarly for the other values.

## Telephony SMS

### Telephony SMS Data Structures

---

advancedParams	Advanced message information for GSM, CDMA, or TDMA messages. This is one of the following structure types: <a href="#">TelSmsDeliveryAdvancedGSMTyp</a> <a href="#">TelSmsDeliveryAdvancedCDMATyp</a> <a href="#">TelSmsDeliveryAdvancedTDMATyp</a>
<-> extensionsCount	On input, this is the number of extension structures allocated for this message. You must allocate at least one structure to specify the multipart information.
	Upon return, this is the number of extensions in the SMS header. If the SMS header contains more extensions than you have allocated, the available extension structures are filled, and this function generates a <code>telErrBufferSize</code> error.
<-> extensionsP	A pointer to an array of <a href="#">TelSmsExtensionType</a> structures that you have allocated for this message. You must allocate this array before using this structure.

## TelSmsExtensionType

The `TelSmsExtensionType` structure specifies multipart information about a message.

```
typedef struct _TelSmsExtensionType
    UInt8      extensionTypeId;
    union
    {
        TelSmsMultiPartExtensionType      mp;
        TelSmsNbsExtensionType          nbs;
        TelSmsUserExtensionType         user;
    } extension;
} TelSmsExtensionType
```

### Field Descriptions

<-> extensionTypeId	Identifies the type of SMS extension structure found in the extension union. This is one of the <a href="#">SMS Extension Type Constants</a> .  If the value of this field is kTelSmsMultipartExtensionTypeId, then the extension union contains a <a href="#">TelSmsMultiPartExtensionType</a> structure.  If the value of this field is kTelSmsNbsExtensionTypeId or kTelSmsNbs2ExtensionTypeId, then the union contains a <a href="#">TelSmsNbsExtensionType</a> structure. The difference between these two is that kTelSmsNbs2ExtensionTypeId indicates that the port value is a long instead of a short.  If this field contains any other value, then the union contains a <a href="#">TelSmsUserExtensionType</a> structure.
<- extension	The extension information, which is one of the following structure types: <a href="#">TelSmsMultiPartExtensionType</a> <a href="#">TelSmsNbsExtensionType</a> <a href="#">TelSmsUserExtensionType</a>

## TelSmsGetAvailableStorageType

The [TelSmsGetAvailableStorage](#) function uses a TelSmsGetAvailableStorageType structure to retrieve information about the storage available on the phone.

```
typedef struct _TelSmsGetAvailableStorageType
{
    UInt16    count;
    UInt8    *storagesP;
} TelSmsGetAvailableStorageType
```

## Telephony SMS

### Telephony SMS Data Structures

---

#### Field Descriptions

- <-> count      The size of the `storagesP` buffer.  
When the structure is used as an input parameter, this is the allocated number of values in the buffer. Upon return, this is the total number of storage areas in the phone.
- <- storagesP      A buffer into which the retrieved storage IDs are stored. Each value stored into the buffer is one of the [SMS Storage ID Constants](#).

## TelSmsGetMessageCountType

The [TelSmsGetMessageCount](#) function uses a `TelSmsGetMessageCountType` structure to retrieve information about messages in the currently selected storage.

```
typedef struct _TelSmsGetMessageCountType
{
    UInt8      messageType;
    UInt16     slots;
    UInt16     count;
} TelSmsGetMessageCountType
```

#### Field Descriptions

- > messageType      The type of message for which you want to retrieve the count. This must be one of the [SMS Message Type Constants](#).  
You must fill this in on input to the [TelSmsGetMessageCount](#) function.
- <- slots      The total number of message slots available in the currently selected storage area (for all message types).
- <- count      The number of filled slots for the specified `messageType` in the currently selected storage area.

## **TelSmsManualAckType**

The [TelSmsSendManualAcknowledge](#) function uses a TelSmsManualAckType structure to specify the information used to send a message acknowledgment.

```
typedef struct _TelSmsManualAckType
{
    UInt16      version;
    Char        *destinationAddress;
    UInt32      messageId;
    UInt16      dataSize;
    UInt8       *data;
    UInt8      dataCodingScheme;
    UInt8      responseCode;
} TelSmsManualAckType
```

### **Field Descriptions**

version	The version of the SMS API used for this acknowledgment.
destinationAddress	The destination address. For GSM, the length of this address is 12 bytes. For CDMA, the length is up to 128 bytes.
messageId	Upon return, the ID of the acknowledgment.
dataSize	The size of the data buffer.  When the structure is used as an input parameter, this is the allocated size of the data buffer.
	Upon return, this is the actual size of the data. If the buffer is too small to contain all of the data, this field is assigned the entire length of the data, and the function using this structure generates a <code>telErrBufferSize</code> error.
data	A buffer into which the retrieved data is stored.  Note that if this buffer is too small to contain all of the retrieved data, the data is truncated and the function using this structure generates a <code>telErrBufferSize</code> error.

## Telephony SMS

### Telephony SMS Data Structures

---

dataCodingScheme	The coding scheme used for the data. This must be one of the <a href="#">SMS Data Coding Scheme Constants</a> .
responseCode	The response code. The value of this field depends on the network being used.

## TelSmsMultiPartExtensionType

The [TelSmsExtensionType](#) structure uses a TelSmsMultiPartExtensionType structure to describe information about a multipart message.

```
typedef struct _TelSmsMultiPartExtensionType
{
    UInt16 bytesSent;
    UInt16 partCurrent;
    UInt16 partCount;
    UInt16 partId;
} TelSmsMultiPartExtensionType
```

### Field Descriptions

<-> bytesSent	On input, set this value to 0.  Upon return, this is the current count of message bytes that have been sent.
<-> partCurrent	On input, set this value to 0.  Upon return, this is the part number of the current message part.

<-> partCount	On input, set this value to 0.  Upon return, this is the number of message parts required to send the data.
<-> partId	The ID of the current SMS message. This ID is unique and is the same for all parts of the message. This information is required to reassemble a multi-part SMS.  On input, set this value to 0.

## **TelSmsNbsExtensionType**

The [TelSmsExtensionType](#) structure uses a TelSmsNbsExtensionType structure to describe information about a NBS message.

```
typedef struct _TelSmsNbsExtensionType
{
    UInt16 destPort;
    UInt16 srcPort;
} TelSmsNbsExtensionType
```

### **Field Descriptions**

<-> destPort	When the structure is used for input, this is the NBS port number used to encode the data.  Upon return, this is the NBS port number that was used for the data.
<-> srcPort	This is currently the same as the destPort.

## **TelSmsReadMessagesType**

The [TelSmsReadMessages](#) function uses a TelSmsReadMessagesType structure to retrieve messages from the currently selected storage area.

```
typedef struct _TelSmsReadMessagesType
{
    UInt16 first;
    UInt16 count;
    TelSmsDeliveryMessageType *messagesP;
}
```

## Telephony SMS

### Telephony SMS Data Structures

---

```
} TelSmsReadMessagesType
```

#### Field Descriptions

- > first      The index of the first message to retrieve. Message indexes are zero-based.
- <-> count      The size of the messagesP buffer.
- When the structure is used as an input parameter, this is the allocated number of pointers in the messagesP buffer.
- Upon return, this is the number of messages that could be retrieved. If the messagesP buffer is too small to contain all of the messages, this field is assigned the entire count, and the function using this structure generates a telErrBufferSize error.
- <-> messagesP      An array of pointers to [TelSmsDeliveryMessageType](#) structures, each of which is filled in with a retrieved message, if available.

## TelSmsReadReportsType

The [TelSmsReadReports](#) function uses a TelSmsReadReportsType structure to retrieve reports from the currently selected storage area.

```
typedef struct _TelSmsReadReportsType
{
    UInt16           first;
    UInt16           count;
    TelSmsReportType *reportsP;
} TelSmsReadReportsType
```

### Field Descriptions

-> first	The index of the first report to retrieve. Report indexes are zero-based.
<-> count	The size of the reportsP buffer.  When the structure is used as an input parameter, this is the allocated number of pointers in the reportsP buffer.
	Upon return, this is the actual number of reports that could be read. If the buffer is too small to contain all of the reports, this field is assigned the entire count, and the function using this structure generates a telErrBufferSize error.
<-> reportsP	An array of pointers to <a href="#">TelSmsReportType</a> structures, each of which is filled in with a retrieved message, if available.  Note that if this buffer is too small to contain all of the retrieved reports, the function using this structure generates a telErrBufferSize error.

## TelSmsReadSubmittedMessagesType

The [TelSmsReadSubmittedMessages](#) function uses a TelSmsReadSubmittedMessagesType structure to retrieve submitted messages from the currently selected storage area.

```
typedef struct _TelSmsReadMessagesType
{
    UInt16           first;
    UInt16           count;
    TelSmsDeliveryMessageType *submittedsP;
} TelSmsReadSubmittedMessagesType
```

## Telephony SMS

### Telephony SMS Data Structures

---

#### Field Descriptions

- > first The index of the first message to retrieve. Message indexes are zero-based.
- <-> count The size of the submittedSP buffer.  
  
When the structure is used as an input parameter, this is the allocated number of pointers in the submittedSP buffer. Upon return, this is the number of messages that were actually read.
- <-> submittedSP An array of pointers to [TelSmsSubmittedMessageType](#) structures, each of which is filled in with a retrieved message, if available.

## TelSmsReportType

The [TelSmsReadReport](#) function uses a TelSmsReportType structure to retrieve a report from the report storage area.

```
typedef struct _TelSmsReportType
{
    UInt16           version;
    UInt16           index;
    UInt8            reportType;
    UInt32           messageId;
    UInt16           dataSize;
    UInt8            *data;
    UInt8            dataCodingScheme;
    Char             *originatingAddress;
    UInt8            originatingAddressSize;
    UInt8            report;
    TelSmsDateTimeType   timeStamp;
} TelSmsReportType
```

#### Field Descriptions

- > version
- <- index The index of the report in the phone storage area.
- <- reportType The delivery report type.

<- messageId	The message ID.
<-> dataSize	The size of the data buffer.  When the structure is used as an input parameter, this is the allocated size of the data buffer.
	Upon return, this is the actual size of the data. If the buffer is too small to contain all of the data, this field is assigned the entire length of the data, and the function using this structure generates a <code>telErrBufferSize</code> error.
<-> data	A buffer into which the retrieved data is stored.  Note that if this buffer is too small to contain the all of the retrieved data, the data is truncated and the function using this structure generates a <code>telErrBufferSize</code> error.
<- dataCodingScheme	The encoding scheme used for the report data. This must be one of the <a href="#">SMS Data Coding Scheme Constants</a> .
<-> originatingAddress	A buffer into which the originating address is stored.  Note that if this buffer is too small to contain the entire retrieved string, the string is truncated (and ends with the null terminator character) and the function using this structure generates a <code>telErrBuffrSize</code> error.
<-> originatingAddressSize	The size of the originatingAddress buffer.  Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a <code>telErrBufferSize</code> error.

## Telephony SMS

### Telephony SMS Data Structures

---

<- report	The ID of the delivery confirmation report associated with the message.
<- timeStamp	An <a href="#">TelSmsDateTimeType</a> structure that stores the time at which the message corresponding to the report was received.

## TelSmsSendMessageType

```
typedef struct _TelSmsSendMessageType
{
    UInt32                     messageId;
    TelSmsSubmitMessageType    message;
} TelSmsSendMessageType
```

### Field Descriptions

<- messageId	The SMS ID that was assigned by the telephone to the outgoing message.
	For a multi-part message, each part has its own message ID.
-> message	A structure of type <a href="#">TelSmsSubmitMessageType</a> that contains the message data and parameters.

## TelSmsSubmitAdvancedCDMAType

The [TelSmsSubmitMessageType](#) structure includes a [TelSmsSubmitAdvancedCDMAType](#) structure for CDMA messages.

```
typedef struct _TelSmsSubmitAdvancedCDMAType
{
    Boolean                  manualAckRequest;
    UInt8                    messageType;
    TelSmsDateTimeType      deferredDate;
    UInt8                   priority;
    UInt8                   privacy;
    Boolean                 alertOnDeliveryRequest;
    Char                    *callbackNumber;
    UInt8                   callbackNumberSize;
```

```
} TelSmsSubmitAdvancedCDMAType
```

### Field Descriptions

manualAckRequest	true if a reply is requested from the user, and false if not.
messageType	The type of the message. This must be one of the <a href="#">SMS Message Type Constants</a> .
deferredDate	
priority	
privacy	The privacy type of the message. This must be one of the <a href="#">SMS Message Privacy Constants</a> .
alertOnDeliveryRequest	true if the user is to be alerted upon delivery of this message, and false if not.
callbackNumber	A buffer into which the retrieved callback telephone number string is stored.  Note that if this buffer is too small to contain the entire retrieved string, the string is truncated (and ends with the null terminator character) and the function using this structure generates a <code>telErrBufferSize</code> error.
callbackNumberSize	When the structure is used as an input parameter, this is the allocated size, in bytes, of the <code>callbackNumber</code> string.  Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a <code>telErrBufferSize</code> error.

## TelSmsSubmitAdvancedGSMTyp

The [TelSmsSubmitMessageType](#) structure includes a `TelSmsSubmitAdvancedGSMTyp` structure for GSM messages.

## Telephony SMS

### Telephony SMS Data Structures

---

```
typedef struct _TelSmsSubmitAdvancedGSMTyp
{
    UInt16      protocolId;
    Boolean     rejectDuplicatedRequest;
    Boolean     replyPath;
    Char        *serviceCenterNumber;
    UInt8       serviceCenterNumberSize;
} TelSmsSubmitAdvancedGSMTyp
```

#### Field Descriptions

- > **protocolId**      Specifies gateway information for routing a message to another transport.  
  
Some service centers provide a gateway between SMS and other transports such as mail and FAX. Service centers may reject messages with **protocolId** values that are reserved or unsupported.
  
- > **rejectDuplicatedRequest**      A Boolean value that specifies if the service center should accept a submit message for a submit message that is still held in the service center when that message has the same identifier and destination address as a previously submitted message from the same originating address.
  
- > **replyPath**      The path that the service center can use to deliver a reply to the originating message.  
  
The reply path is requested by the originating mobile device by setting the **replyPath** parameter in the original submit message.
  
- If the service center supports reply path requests from the mobile device, the service center sets the **replyPath** parameter in the response.

- > serviceCenterNumber A buffer containing the service telephone number string.
- > serviceCenterNumberSize The allocated size, in bytes, of the serviceCenterNumber string.

## **TelSmsSubmitAdvancedTDMAType**

The [TelSmsSubmitMessageType](#) structure includes a TelSmsSubmitAdvancedTDMAType structure for TDMA messages.

```
typedef struct _TelSmsSubmitAdvancedTDMAType
{
    Boolean           manualAckRequest;
    UInt8             messageType;
    TelSmsDateTimeType deferredDate;
    UInt8             priority;
    UInt8             privacy;
    Boolean           alertOnDeliveryRequest;
    Char              *callbackNumber;
    UInt8             callbackNumberSize;
} TelSmsSubmitAdvancedTDMAType
```

### **Field Descriptions**

manualAckRequest	true if a reply is requested from the user, and false if not.
messageType	The type of the message. This must be one of the <a href="#">SMS Message Type Constants</a> .
deferredDate	
priority	
privacy	The privacy type of the message. This must be one of the <a href="#">SMS Message Privacy Constants</a> .
alertOnDeliveryRequest	true if the user is to be alerted upon delivery of this message, and false if not.

## Telephony SMS

### Telephony SMS Data Structures

---

callbackNumber	A buffer into which the retrieved callback telephone number string is stored.  Note that if this buffer is too small to contain the entire retrieved string, the string is truncated (and ends with the null terminator character) and the function using this structure generates a <code>telErrBufferSize</code> error.
callbackNumberSize	When the structure is used as an input parameter, this is the allocated size, in bytes, of the <code>callbackNumber</code> string.  Upon return, this is the actual size of the string, including the null terminator character. If the buffer is too small to contain the entire retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a <code>telErrBufferSize</code> error.

## TelSmsSubmitMessageType

The [TelSmsReadReports](#) and [TelSmsSendMessageType](#) structures use a `TelSmsSubmitMessageType` structure to stored reports retrieved from the currently selected storage area.

```
typedef struct _TelSmsSubmitMessageType
{
    UInt16           version;
    Boolean          networkDeliveryRequest;
    Char             *destinationAddress;
    UInt8            destinationAddressSize;
    UInt16           dataSize;
    UInt8            *data;
    UInt8            dataCodingScheme;
    TelSmsDateTimeType validityPeriod;
    UInt8            standardType;
    union
    {
        TelSmsSubmitAdvancedGSMTyp... advancedGSM;
        TelSmsSubmitAdvancedCDMATyp... advancedCDMA;
    }
}
```

```

        TelSmsSubmitAdvancedTDMAType advancedTDMA;
    } advancedParams;
    UInt8                     extensionsCount;
    TelSmsExtensionType      *extensionsP;
} TelSmsSubmitMessageType

```

### Field Descriptions

-> version	The version of the SMS API associated with this message.
-> networkDeliveryRequest	If this value is true, the service center accepts the submit message.
<-> destinationAddress	A buffer that contains the phone number of the message recipient.
<-> destinationAddressSize	The size of the destination address string.
<-> dataSize	The size of the data buffer.
<-> data	A buffer into which the retrieved message data is stored.
-> dataCodingScheme	The coding scheme used for the data. This is one of the <a href="#">SMS Data Coding Scheme Constants</a> .
-> validityPeriod	An <a href="#">TelSmsDateTimeType</a> structure that specifies the amount of time for which the message is valid.
-> standardType	Indicates which field of the advancedParams union contains the message information. This must be one of the <a href="#">Network Type Constants</a> described in <a href="#">Chapter 70, “Telephony Network.”</a>
	If this value is kTelNwkCDMA, then the advancedParams union contains a <a href="#">TelSmsDeliveryAdvancedCDMAType</a> structure, and similarly for the other values.

## Telephony SMS

### Telephony SMS Data Structures

---

-> advancedParams	Advanced message information for GSM, CDMA, or TDMA messages. This is a pointer to one of the following structure types: <a href="#">TelSmsSubmitAdvancedCDMAType</a> <a href="#">TelSmsDeliveryAdvancedGSMTyp</a> e <a href="#">TelSmsDeliveryAdvancedTDMATyp</a> e
<-> extensionsCount	The number of extension structures allocated for this message. You must allocate at least one structure to specify the multipart information.
<-> extensionsP	A pointer to an array of <a href="#">TelSmsExtensionType</a> structures that you have allocated for this message.

## TelSmsSubmittedMessageType

The [TelSmsReadSubmittedMessage](#) function uses a TelSmsSubmittedMessageType structure to retrieve reports from the currently selected storage area.

```
typedef struct _TelSmsSubmittedMessageType
{
    UInt16           index;
    TelSmsSubmitMessageType   message;
} TelSmsSubmittedMessageType
```

### Field Descriptions

-> index	The index of the message on the phone.
<- message	A <a href="#">TelSmsSubmitMessageType</a> structure that represents the message.

## TelSmsUserExtensionType

The [TelSmsExtensionType](#) structure uses a TelSmsUserExtensionType structure to describe a user-defined extended message header.

```
typedef struct _TelSmsUserExtensionType
{
    UInt8      *extHeader;
    UInt8      extHeaderSize;
} TelSmsUserExtensionType
```

### Field Descriptions

<-> extHeader	On input, this field must be set to 0. Upon return, this is a pointer to the user-defined header content.
<-> extHeaderSize	On input, this field must be set to 0. Upon return, this is the size of the user-defined header content.

## Telephony SMS Constants

This section describes the constants used with the SMS service set of the telephony API.

### SMS Extension Type Constants

The SMS extension type constants describe the type of extension used to represent part of a message.

Constant	Value	Description
kTelSmsMultiPartExtensionTypeId	0x00	A multipart short message.
kTelSmsNbsExtensionTypeId	0x04	An NBS message with short port number value.
kTelSmsNbs2ExtensionTypeId	0x05	An NBS message with long port number value.
(any other value)		A user-defined extension type.

### SMS Message Type Constants

The SMS message type constants describe the delivery type of a message.

## Telephony SMS

### Telephony SMS Constants

---

Constant	Value	Description
kTelSmsMessageTypeDelivered	0	A delivered message.
kTelSmsMessageTypeReport	1	A report message.
kTelSmsMessageTypeSubmitted	2	A submitted message.
kTelSmsMessageTypeManualAck	3	A manual acknowledgement message.
kTelSmsMessageTypeAllTypes	4	All messages.

## SMS Message Transport Protocol Constants

The SMS message transport protocol constants describe the protocol used to deliver a message.

Constant	Value	Description
ktelSmsDefaultProtocol	0	The default message transport protocol.
kTelSmsFaxProtocol	1	A FAX message.
kTelSmsX400Protocol	2	An X.400 message.
kTelSmsPagingProtocol	3	A paged message.
kTelSmsEmailProtocol	4	An email message.
kTelSmsErmesProtocol	5	An Ermes message.
kTelSmsVoiceProtocol	6	A voice message.

## SMS Storage ID Constants

The SMS storage ID constants describe the storage location of a message.

Constant	Value	Description
kTelSmsStorageSIM	0	Stored in the SIM.
kTelSmsStoragePhone	1	Stored in the phone.
kTelSmsStorageAdaptor	2	Stored in the telephone adaptor.
kTelSmsStorageFirstOem	3	Storage managed by the OEM.
		This constant specifies the first OEM storage area. You can specify additional OEM storage areas by incrementing this value. For example, to specify the third OEM storage area, use the following:
		<code>kTelSmsStorageFirstOem+2</code>

## SMS Data Coding Scheme Constants

The SMS data coding scheme constants describe the encoding used for SMS data.

Constant	Value	Description
kTelSms8BitsEncoding	0	8-bit encoding.
kTelSmsBitsASCIIEncoding	1	ANSI X3.4 encoding.
kTelSmsIA5Encoding	2	CCITT T.50 encoding.
kTelSmsIS91Encoding	3	TIA/EIA/IS-91 section 3.7.1 encoding.
kTelSmsUCS2Encoding	4	UCS2 encoding; used with GSM only.
kTelSmsDefaultGSMEncoding	5	Default encoding for GSM only.

## Telephony SMS

### Telephony SMS Functions

---

## SMS Message Urgency Constants

The SMS message urgency constants describe the priority level of a message in a [TelSmsDeliveryAdvancedCDMAType](#) or [TelSmsDeliveryAdvancedTDMAType](#) structure.

Constant	Value	Description
kTelSmsUrgencyNormal	0	Normal urgency.
kTelSmsUrgencyUrgent	1	An urgent message.
kTelSmsUrgencyEmergency	2	An emergency message.

## SMS Message Privacy Constants

The SMS message privacy constants describe the privacy type of a message in a CDMA or TDMA advanced parameters.

Constant	Value	Description
kTelSmsPrivacyNotRestricted	0	Privacy level 0.
kTelSmsPrivacyRestricted	1	Privacy level 1.
kTelSmsPrivacyConfidential	2	Privacy level 2.
kTelSmsPrivacySecret	3	Privacy level 3.

## Telephony SMS Functions

This section describes the functions used with the SMS service set of the telephony API.

## TelSmsDeleteMessage

<b>Purpose</b>	Delete an SMS report, delivered message, or submitted message.								
<b>Declared In</b>	TelephonyMgr.h								
<b>Prototype</b>	<pre>Err TelSmsDeleteMessage (UInt16 iRefnum, TelAppID iAppId, TelSmsDeleteMessageType *ioParamP, UInt16 *ioTransIdP)</pre>								
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>&lt;-&gt; ioParamP</td><td>A pointer to a <a href="#">TelSmsDeleteMessageType</a> structure that specifies the index and type of the message to delete.</td></tr><tr><td>&lt;-&gt; ioTransIdP</td><td>Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</td></tr></table>	-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	<-> ioParamP	A pointer to a <a href="#">TelSmsDeleteMessageType</a> structure that specifies the index and type of the message to delete.	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
-> iRefnum	The telephony manager library reference number.								
-> iAppId	The telephone application attachment identifier for your application.								
<-> ioParamP	A pointer to a <a href="#">TelSmsDeleteMessageType</a> structure that specifies the index and type of the message to delete.								
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.								
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.								
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes: <table><tr><td>returnCode</td><td>errNone upon success or an error code upon failure.</td></tr><tr><td>transId</td><td>The transaction ID of the operation.</td></tr></table>	returnCode	errNone upon success or an error code upon failure.	transId	The transaction ID of the operation.				
returnCode	errNone upon success or an error code upon failure.								
transId	The transaction ID of the operation.								

## Telephony SMS

### Telephony SMS Functions

---

paramP	Points to the <a href="#">TelSmsDeleteMessageType</a> structure passed to this function in the iDelInfoP parameter.
functionId	kTelSmsDeleteMessageMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by ioParamP remains in memory until the asynchronous call completes.

---

<b>Comments</b>	If the deleted message has been delivered, the deletion is performed in the current storage, which you can set with the <a href="#">TelSmsSelectStorage</a> function.  Before using this function, you should verify that it is available by calling the <a href="#">TelIsSmsServiceAvailable</a> macro.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelSmsSelectStorage</a>

## TelSmsGetAvailableStorage

<b>Purpose</b>	Retrieve the list of all available storage on the phone.	
<b>Declared In</b>	TelephonyMgr.h	
<b>Prototype</b>	Err TelSmsGetAvailableStorage(UInt16 iRefnum, TelAppID iAppId, TelSmsGetAvailableStorageType *ioParamP, UInt16 *ioTransIdP)	
<b>Parameters</b>	-> iRefnum	The telephony manager library reference number.
	-> iAppId	The telephone application attachment identifier for your application.

<-> `ioParamP` A pointer to a [`TelSmsGetAvailableStorageType`](#) structure that is filled in with information about the storage areas available on the phone.

<-> `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [`TelEventType`](#) event that is sent when the operation completes:

`returnCode` `errNone` upon success or an error code upon failure.

`transId` The transaction ID of the operation.

`paramP` Points to the [`TelSmsGetAvailableStorageType`](#) structure passed to this function in the `ioParamP` parameter.

`functionId` `kTelSmsGetAvailableStorageMessage`

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

---

**Comments** The count of storage areas available on the phone is stored into the `count` field of the [`TelSmsGetAvailableStorageType`](#) structure referenced by `ioParamP`, and the storage ID of each available room is stored into the buffer referenced by the `storagesP` field. If the `storagesP` buffer is too small to contain all of the storage IDs, the buffer is truncated and this function returns the

## Telephony SMS

### Telephony SMS Functions

---

`telErrBufferSize` error. The count field of the structure is always updated to contain the total number of available storage areas on the phone.

Before using this function, you should verify that it is available by calling the [TelIsSmsServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSmsSelectStorage](#)

## TelSmsGetDataMaxSize

**Purpose** Returns the maximum length, in bytes, of a message on the current network.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelSmsGetDataMaxSize(UInt16 iRefnum,  
TelAppID iAppId, UInt16 *oSizeP,  
UInt16 *ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
<- <code>oSizeP</code>	A pointer to an unsigned integer value that is updated with the maximum length of an SMS message on the current network.
<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode      `errNone` upon success or an error code upon failure.

transId          The transaction ID of the operation.

paramP          Points to the unsigned integer value passed to this function in the `oSizeP` parameter.

functionId      `kTelGetDataMaxSizeMessage`

---

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by `oSizeP` remains in memory until the asynchronous call completes.

---

**Comments** You can use this function to determine the maximum size you need to allocate to read a message.

Before using this function, you should verify that it is available by calling the [TelIsSmsServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSmsReadMessage](#), [TelSmsReadMessages](#)

## Telephony SMS

### Telephony SMS Functions

---

## TelSmsGetMessageCount

**Purpose** Retrieve the total number of message slots, and the number of filled slots for the specified message type.

**Declared In** `TelephonyMgr.h`

**Prototype**

```
Err TelSmsGetMessageCount (UInt16 iRefnum,
                           TelAppID iAppId,
                           TelSmsGetMessageCountType *ioParamP,
                           UInt16 *ioTransIdP)
```

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
<-> <code>ioParamP</code>	A pointer to a <a href="#">TelSmsGetMessageCountType</a> structure that specifies the message type and is filled in with the count information.
<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> upon success or an error code upon failure.
<code>transId</code>	The transaction ID of the operation.

paramP Points to the [TelSmsGetMessageCountType](#) structure passed to this function in the ioParamP parameter.

functionId kTelSmsGetMessageCountMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

---

**Comments**

The currently selected storage area pertains only to delivered messages; other message types are stored on the phone, but not in a specific storage area. If you specify delivered messages, this function retrieves information about the messages in the currently selected SMS storage area on the phone. If you specify a different message type, this function retrieves information about the messages in the phone.

You specify the message type by assigning one of the [SMS Message Type Constants](#) to the `messageType` field of the `TelSmsGetMessageCountType` structure before calling this function.

Before using this function, you should verify that it is available by calling the [TelIsSmsServiceAvailable](#) macro.

**Compatibility**

Implemented only if [4.0 New Feature Set](#) is present.

**See Also**

[TelSmsSelectStorage](#)

## Telephony SMS

### Telephony SMS Functions

---

## TelSmsGetSelectedStorage

**Purpose** Retrieve the ID of the currently selected storage area on the phone.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelSmsGetSelectedStorage(UInt16 iRefnum,  
TelAppID iAppId, UInt8 *oStorageIdP,  
UInt16 *ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
<- <code>oStorageIdP</code>	A pointer to an unsigned byte value that is assigned the ID of the currently selected storage area on the phone. The assigned ID value is one of the <a href="#">SMS Storage ID Constants</a> .
<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

<code>returnCode</code>	<code>errNone</code> upon success or an error code upon failure.
<code>transId</code>	The transaction ID of the operation.

paramP	Points to the unsigned integer value passed to this function in the <code>oStorageIdP</code> parameter.
functionId	<code>kTelGetSelectedStorageMessage</code>

---

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by `oStorageIdP` remains in memory until the asynchronous call completes.

---

**Comments** The currently selected storage area pertains only to delivered messages; other message types are stored on the phone, but not in a specific storage area.  
Before using this function, you should verify that it is available by calling the [TelIsSmsServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSmsSelectStorage](#)

## TelSmsGetUniquePartId

**Purpose** Return a unique part identifier to assign to the `partId` field of a submit message.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelSmsGetUniquePartId(UInt16 iRefnum,  
TelAppID iAppId, UInt16 *oUniqueIdP,  
UInt16 *ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
<- <code>oUniqueIdP</code>	A pointer to a unsigned integer value. Upon return, this is the unique part ID value.

## Telephony SMS

### Telephony SMS Functions

---

<-> `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

`returnCode` `errNone` upon success or an error code upon failure.

`transId` The transaction ID of the operation.

`paramP` Points to the unsigned integer value passed to this function in the `oUniqueIdP` parameter.

`functionId` `kTelGetUniqueIdMessage`

---

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by `oUniqueIdP` remains in memory until the asynchronous call completes.

---

**Comments** This function corresponds to the `kTelUrgSmsGetUniqueIdMessage` value.

Before using this function, you should verify that it is available by calling the [TelIsSmsServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSmsSendMessage](#)

## TelSmsReadMessage

**Purpose** Retrieve a delivered message from the currently selected storage area.

**Declared In** `TelephonyMgr.h`

**Prototype**

```
Err TelSmsReadMessage(UInt16 iRefnum,
                      TelAppID iAppId,
                      TelSmsDeliveryMessageType *ioMessageP,
                      UInt16 *ioTransIdP)
```

<b>Parameters</b>	-> <code>iRefnum</code>	The telephony manager library reference number.
	-> <code>iAppId</code>	The telephone application attachment identifier for your application.
	<-> <code>ioMessageP</code>	A pointer to a <code>TelSmsDeliveryMessageType</code> structure that is filled in with the message.  On input, the <code>index</code> field of this structure contains the index of the message that you want retrieved. Message indexes are zero-based.
		On input, the <code>extensionsCount</code> field specifies the number of extensions allocated in the <code>extensionsP</code> array. You must allocate at least one multi-part extension, even for a single-part message.
		On input, the <code>dataSize</code> field specifies the allocated size of the data buffer, and the <code>originatingAddressSize</code> field specifies the allocated size of the <code>originatingAddress</code> string. Upon return, each size field specifies the complete size of the data that was stored into the buffer, even if the data had to be truncated to fit.

## Telephony SMS

### Telephony SMS Functions

---

<-> `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

#### Synchronous Result

Returns `errNone` if the function was successful or returns an error code if not successful.

#### Asynchronous Result

The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

`returnCode`      `errNone` upon success or an error code upon failure.

`transId`      The transaction ID of the operation.

`paramP`      Points to the [TelSmsDeliveryMessageType](#) structure passed to this function in the `ioMessageP` parameter.

`functionId`      `kTelSmsReadMessageMessage`

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioMessageP` remains in memory until the asynchronous call completes.

---

#### Comments

The message data is stored into the `data` field of the [TelSmsDeliveryMessageType](#) structure referenced by `ioMessageP`. If the complete message data is too large to fit into the allocated size of the `data` field, the message data is truncated and this function returns the `telErrBufferSize` error. The `dataSize` field of the structure is always updated to contain the actual size, in bytes, of the message data.

The originating address string is stored into the `originatingData` field of the [TelSmsDeliveryMessageType](#) structure referenced by `ioMessageP`. If the complete string is too large to fit into the allocated size of the `originatingData` field,

the string is truncated (and ends with the null terminator character) and this function returns the `telErrBufferSize` error. The `originatingAddressSize` field of the structure is always updated to contain the actual size of the string.

Before calling this function, you need to allocate a number of fields and structures in and related to the `TelSmsDeliveryMessage` structure:

- Allocate each address field with a size of at least `kTelMaxPhoneNumberLen + 1`.

For example, for a GSM message, you must allocate the `originatingAddress` and `serviceCenterNumber` fields in the `TelSmsDeliveryAdvancedGSMType` structure in the `TelSmsDeliveryMessage` structure.

- Allocate the message field data to have the maximum size of a message on the current network. You can determine this value by calling the `TelSmsGetDataMaxSize` function.
- Allocate at least one `TelSmsExtensionType` structure in the `TelSmsDeliveryMessage` structure. You must have at least one extension structure, even if your message has only a single part. If you do not allocate enough extensions for the message, `TelSmsReadMessage` returns an error. Palm recommends allocating between 3 and 5 extensions for a message.
- You should not allocate a pointer for user extension data. If you receive user extension data, the user extension pointer will reference a block in the message data. Do not deallocate the user extension data when you release the structure.

Before using this function, you should verify that it is available by calling the `TelIsSmsServiceAvailable` macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSmsSelectStorage](#)

## TelSmsReadMessages

**Purpose** Retrieve a range of delivered messages from the currently selected storage area.

**Declared In** `TelephonyMgr.h`

**Prototype**

```
Err TelSmsReadMessages (UInt16 iRefnum,
                        TelAppID iAppId,
                        TelSmsReadMessagesType *ioParamP,
                        UInt16 *ioTransIdP)
```

<b>Parameters</b>	-> <code>iRefnum</code>	The telephony manager library reference number.
	-> <code>iAppId</code>	The telephone application attachment identifier for your application.
	<-> <code>ioParamP</code>	A pointer to a <a href="#">TelSmsReadMessagesType</a> structure.  On input, the <code>first</code> field of this structure specifies the index of the first message to retrieve. Message indexes are zero-based.
		On input the <code>count</code> field of this structure specifies the allocated size of the <code>messagesP</code> buffer. Upon return, the <code>count</code> field specifies the actual number of messages that were available, even if that many could not fit into the buffer.
		The <code>messagesP</code> buffer must contain pointers to <a href="#">TelSmsDeliveryMessageType</a> structures that have been allocated. Each of these structures must be initialized as described for the <a href="#">TelSmsReadMessage</a> function.
	<-> <code>ioTransIdP</code>	Set the value of this parameter to <code>NULL</code> to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

`returnCode`      `errNone` upon success or an error code upon failure.

`transId`      The transaction ID of the operation.

`paramP`      Points to the [TelSmsReadMessagesType](#) structure passed to this function in the `ioParamP` parameter.

`functionId`      `kTelSmsReadMessagesMessage`

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` and all of the structures referenced by it remain in memory until the asynchronous call completes.

---

**Comments** If the message data or originating address string data for any of the retrieved messages is larger than the allocated size of its corresponding buffer, the data is truncated into the buffer, and this function returns the `telErrBufferSize` error.

For more information about using this function and allocating structures for its use, see the Comments description for the [TelSmsReadMessage](#) function.

Before using this function, you should verify that it is available by calling the [TelIsSmsServiceAvailable](#) macro.

## Telephony SMS

### Telephony SMS Functions

---

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSmsSelectStorage](#)

## TelSmsReadReport

**Purpose** Read a report from the currently selected storage area.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelSmsReadReport (UInt16 iRefnum,  
TelAppID iAppId, TelSmsReportType *ioReportP,  
UInt16 *ioTransIdP)`

<b>Parameters</b>	<code>-&gt; iRefnum</code>	The telephony manager library reference number.
	<code>-&gt; iAppId</code>	The telephone application attachment identifier for your application.
	<code>&lt;-&gt; ioReportP</code>	A pointer to a <a href="#">TelSmsReportType</a> structure. On input the index field of this structure contains the index of the message that you want retrieved. Message indexes are zero-based.
		On input, the dataSize field specifies the allocated size of the data buffer, and the originatingAddressSize field specifies the allocated size of the originatingAddress string. Upon return, each size field specifies the complete size of the data that was stored into the buffer, even the data had to be truncated to fit.
	<code>&lt;-&gt; ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:  returnCode      errNone upon success or an error code upon failure.  transId          The transaction ID of the operation.  paramP          Points to the <a href="#">TelSmsReportType</a> structure passed to this function in the ioReportP parameter.  functionId      kTelSmsReadReportMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by iEntryP remains in memory until the asynchronous call completes.

---

<b>Comments</b>	<p>The report data is stored into the data field of the <a href="#">TelSmsReportType</a> structure referenced by ioReportP. If the complete message data is too large to fit into the allocated size of the data field, the report data is truncated and this function returns the telErrBufferSize error. The dataSize field of the structure is always updated to contain the actual size, in bytes, of the report data.</p> <p>The originating address string is stored into the originatingData field of the <a href="#">TelSmsReportType</a> structure referenced by ioReportP. If the complete string is too large to fit into the allocated size of the originatingData field, the string is truncated (and ends with the null terminator character) and this function returns the telErrBufferSize error. The originatingAddressSize field of the structure is always updated to contain the actual size of the string.</p> <p>Before using this function, you should verify that it is available by calling the <a href="#">TelIsSmsServiceAvailable</a> macro.</p>
-----------------	--

## Telephony SMS

### Telephony SMS Functions

---

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSmsSelectStorage](#), [TelSmsSendMessage](#)

## TelSmsReadReports

**Purpose** Retrieve a range of reports from the currently selected storage.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelSmsReadReports(UInt16 iRefnum,  
TelAppID iAppId, TelSmsReadReportsType *ioParamP,  
UInt16 *ioTransIdP)`

<b>Parameters</b>	<code>-&gt; iRefnum</code>	The telephony manager library reference number.
	<code>-&gt; iAppId</code>	The telephone application attachment identifier for your application.
	<code>&lt;-&gt; ioParamP</code>	A pointer to a <a href="#">TelSmsReadReportsType</a> structure.  On input, the <code>first</code> field of this structure specifies the index of the first report to retrieve. Report indexes are zero-based.
	<code>&lt;-&gt; ioTransIdP</code>	On input the <code>count</code> field of this structure specifies the allocated size of the <code>reportsP</code> buffer. Upon return, the <code>count</code> field specifies the actual number of reports that were available, even if that many could not fit into the buffer.  The <code>reportsP</code> buffer must contain pointers to <a href="#">TelSmsReportType</a> structures that have been allocated. Each of these structures must be initialized as described for the <a href="#">TelSmsReadReport</a> function.
		<code>&lt;-&gt; ioTransIdP</code> Set the value of this parameter to <code>NULL</code> to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode      errNone upon success or an error code upon failure.

transId          The transaction ID of the operation.

paramP           Points to the [TelSmsReadReportsType](#) structure passed to this function in the ioParamP parameter.

functionId       kTelSmsReadReportsMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by ioParamP and all of the structures referenced by it remain in memory until the asynchronous call completes.

---

**Comments** If the report data or originating address string data for any of the retrieved messages is larger than the allocated size of its corresponding buffer, the data is truncated into the buffer, and this function returns the telErrBufferSize error. For more information, see the Comments description for the [TelSmsReadReport](#) function.

Before using this function, you should verify that it is available by calling the [TelIsSmsServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSmsSelectStorage](#), [TelSmsSendMessage](#)

## Telephony SMS

### Telephony SMS Functions

---

## TelSmsReadSubmittedMessage

**Purpose** Read a previously submitted message that was kept on the phone after being sent.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelSmsReadSubmittedMessage (UInt16 iRefnum,  
TelAppID iAppId,  
TelSmsSubmittedMessageType *ioMessageP,  
UInt16 *ioTransIdP)`

<b>Parameters</b>	<code>-&gt; iRefnum</code>	The telephony manager library reference number.
	<code>-&gt; iAppId</code>	The telephone application attachment identifier for your application.
	<code>&lt;-&gt; ioMessageP</code>	A pointer to a <a href="#">TelSmsSubmitMessageType</a> structure that is filled in with the message.  On input, the index field contains the index of the message that you want retrieved. Message indexes are zero-based.  Upon return, the message field structure is filled in with the message information. You must initialize the buffer size fields of this structure prior to calling this function, including the dataSize, callbackNumberSize, serviceCenterNumberSize, and destinationAddressSize fields. Each of these fields is initialized with the allocated size of its corresponding buffer.
	<code>&lt;-&gt; ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:  returnCode      errNone upon success or an error code upon failure.  transId          The transaction ID of the operation.  paramP          Points to the <a href="#">TelSmsSubmitMessageType</a> structure passed to this function in the ioMessageP parameter.  functionId      kTelSmsSubmittedMessageMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by iEntryP remains in memory until the asynchronous call completes.

---

<b>Comments</b>	<p>The message data is stored into the data field of the <a href="#">TelSmsSubmitMessageType</a> structure referenced by ioMessageP. If the complete message data is too large to fit into the allocated size of the data field, the end of the message data is truncated and this function returns the telErrBufferSize error. The dataSize field of the structure is always updated to contain the actual size, in bytes, of the message data.</p> <p>The same strategy applies to the callbackNumber buffer and callbackNumberSize fields, the destinationAddress buffer and destinationAddressSize fields, and the serviceCenterNumber buffer and serviceCenterNumberSize fields. If the size of the data for any of the buffer fields exceeds the allocated length of the buffer, the end of the data is truncated and this function returns the telErrBufferSize error. Each of the size fields is always updated to contain the complete size of the data intended for the buffer.</p>
-----------------	---

## Telephony SMS

### Telephony SMS Functions

---

Before using this function, you should verify that it is available by calling the [TelIsSmsServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSmsSelectStorage](#), [TelSmsSendMessage](#)

## TelSmsReadSubmittedMessages

**Purpose** Retrieve a range of submitted messages from the currently selected storage area.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelSmsReadSubmittedMessages(UInt16 iRefnum,  
TelAppID iAppId,  
TelSmsReadSubmittedMessagesType *ioParamP,  
UInt16 *ioTransIdP)`

<b>Parameters</b>	-> <code>iRefnum</code>	The telephony manager library reference number.
	-> <code>iAppId</code>	The telephone application attachment identifier for your application.
	<-> <code>ioParamP</code>	A pointer to a <a href="#">TelSmsReadMessagesType</a> structure.  On input, the <code>first</code> field of this structure specifies the index of the first message to retrieve. Message indexes are zero-based.
		On input, the <code>count</code> field of this structure specifies the allocated size of the <code>submittedSP</code> buffer. Upon return, the <code>count</code> field specifies the actual number of messages that were available, even if that many could not fit into the buffer.

The submitted `dsP` buffer must contain pointers to [TelSmsSubmittedMessageType](#) structures that have been allocated. Each of these structures must be initialized as described for the [TelSmsReadSubmittedMessage](#) function.

`<-> ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result**

Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result**

The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

`returnCode`      `errNone` upon success or an error code upon failure.

`transId`      The transaction ID of the operation.

`paramP`      Points to the [TelSmsReadSubmittedMessagesType](#) structure passed to this function in the `ioParamP` parameter.

`functionId`      `kTelSmsReadSubmittedMessagesMessage`

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` and all of the structures referenced by it remain in memory until the asynchronous call completes.

---

**Comments**

If the message data or any of the other variable-length data for any of the retrieved messages is larger than the allocated size of its corresponding buffer, the end of the data is truncated, and this

## Telephony SMS

### Telephony SMS Functions

---

function returns the `telErrBufferSize` error. For more information, see the `Comments` description for the [`TelSmsReadSubmittedMessage`](#) function.

Before using this function, you should verify that it is available by calling the [`TelIsSmsServiceAvailable`](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [`TelSmsSelectStorage`](#), [`TelSmsSendMessage`](#)

## TelSmsSelectStorage

**Purpose** Select a storage area on the phone as the current storage area.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelSmsSelectStorage(UInt16 iRefnum,  
TelAppID iAppId, UInt8 iStorageId,  
UInt16 *ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
-> <code>ioStorageId</code>	The ID of the storage area. This must be one of the <a href="#">SMS Storage ID Constants</a> .
<-> <code>ioTransIdP</code>	Set the value of this parameter to <code>NULL</code> to cause the function to execute synchronously.  If this parameter is not <code>NULL</code> , the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:
returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the unsigned integer value passed to this function in the <code>ioStorageId</code> parameter.
functionId	<code>kTelSmsSelectStorageMessage</code>
<b>Comments</b>	Before using this function, you should verify that it is available by calling the <a href="#">TelIsSmsServiceAvailable</a> macro.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelSmsGetAvailableStorage</a> , <a href="#">TelSmsGetSelectedStorage</a>

## TelSmsSendManualAcknowledge

<b>Purpose</b>	Send a manual acknowledgment of a previously received message. Note that this function is not supported on GSM networks.							
<b>Declared In</b>	<code>TelephonyMgr.h</code>							
<b>Prototype</b>	<pre>Err TelSmsSendManualAcknowledge(UInt16 iRefnum,                                 TelAppID iAppId, TelSmsManualAckType *ioAckP,                                 UInt16 *ioTransIdP)</pre>							
<b>Parameters</b>	<table border="0"> <tr> <td>-&gt; iRefnum</td> <td>The telephony manager library reference number.</td> </tr> <tr> <td>-&gt; iAppId</td> <td>The telephone application attachment identifier for your application.</td> </tr> <tr> <td>&lt;-&gt; ioAckP</td> <td>A pointer to a structure of type <a href="#">TelSmsManualAckType</a>. The fields of this structure specify information about the message being acknowledged.</td> </tr> </table>		-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	<-> ioAckP	A pointer to a structure of type <a href="#">TelSmsManualAckType</a> . The fields of this structure specify information about the message being acknowledged.
-> iRefnum	The telephony manager library reference number.							
-> iAppId	The telephone application attachment identifier for your application.							
<-> ioAckP	A pointer to a structure of type <a href="#">TelSmsManualAckType</a> . The fields of this structure specify information about the message being acknowledged.							

## Telephony SMS

### Telephony SMS Functions

---

Upon return, the messageId field is filled in with the ID of the acknowledgment.

<-> ioTransIdP Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode errNone upon success or an error code upon failure.

transId The transaction ID of the operation.

paramP Points to the [TelSmsManualAckType](#) structure passed to this function in the ioAckP parameter.

functionId kTelSmsSendManualAcknowledgeMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by iEntryP remains in memory until the asynchronous call completes.

---

**Comments** Before using this function, you should verify that it is available by calling the [TelIsSmsServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## TelSmsSendMessage

<b>Purpose</b>	Send an SMS message.								
<b>Declared In</b>	TelephonyMgr.h								
<b>Prototype</b>	<pre>Err TelSmsSendMessage(UINT16 iRefnum,                       TelAppID iAppId,                       TelSmsSendMessageType *ioMessageP,                       UINT16 *ioTransIdP)</pre>								
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>&lt;-&gt; ioMessageP</td><td>A pointer to a structure of type <a href="#">TelSmsSendMessageType</a>. On input, the message field of this structure contains a <a href="#">TelSmsSubmitMessageType</a> with the message to send. You must also allocate and zero at least one <a href="#">TelSmsExtensionType</a> structure for the multi-part information. On output, the messageId field of this structure is filled in with the ID that was assigned to the sent message.</td></tr><tr><td>&lt;-&gt; ioTransIdP</td><td>Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</td></tr></table>	-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	<-> ioMessageP	A pointer to a structure of type <a href="#">TelSmsSendMessageType</a> . On input, the message field of this structure contains a <a href="#">TelSmsSubmitMessageType</a> with the message to send. You must also allocate and zero at least one <a href="#">TelSmsExtensionType</a> structure for the multi-part information. On output, the messageId field of this structure is filled in with the ID that was assigned to the sent message.	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
-> iRefnum	The telephony manager library reference number.								
-> iAppId	The telephone application attachment identifier for your application.								
<-> ioMessageP	A pointer to a structure of type <a href="#">TelSmsSendMessageType</a> . On input, the message field of this structure contains a <a href="#">TelSmsSubmitMessageType</a> with the message to send. You must also allocate and zero at least one <a href="#">TelSmsExtensionType</a> structure for the multi-part information. On output, the messageId field of this structure is filled in with the ID that was assigned to the sent message.								
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.								
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.								
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:								

## Telephony SMS

### Telephony SMS Functions

---

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the <a href="#">TelSmsSendMessageType</a> structure passed to this function in the ioMessageP parameter.
functionId	kTelSmsSendMessageMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioMessageP` remains in memory until the asynchronous call completes.

---

#### Comments

You need to make multiple calls to the [TelSmsSendMessage](#) function to send your message:

- The first call to `TelSmsSendMessage` does not actually send the message. It computes the number of parts and fills in the multi-part extension structures in the `TelSmsSendMessageType` structure. Note that you must allocate at least one extension structure, even for single-part messages.
- Subsequent calls to `TelSmsSendMessage` actually send the data.
- To send an entire message, you need to call `TelSmsSendMessage` in a loop. Terminate the loop when an error occurs, or when the `byteSend` field of the first `TelSmsExtensionType` structure has the same value as the `dataSize` field of the `TelSmsSendMessageType` structure that represents the message. For example:

---

```
while (!TelSmsSendMessage(...) &&
    msg.extensionP[0].extension.mp.byteSend != dataSize);
```

---

Before using this function, you should verify that it is available by calling the [TelIsSmsServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelSmsSelectStorage](#), [TelCfgSetSmsCenter](#)

## **Telephony SMS**

### *Telephony SMS Functions*

---

# Telephony Phone Book

---

This chapter describes the phone book service set of the telephony API.

For more information about the telephony manager basic services and the different service sets, see [Chapter 68, “Telephony Basic Services.”](#)

This chapter describes:

- [Telephony Phone Book Data Structures](#)
- [Telephony Phone Book Constants](#)
- [Telephony Phone Book Functions](#)

For more information about using the telephony manager, see [Chapter 10, “Telephony Manager”](#) in *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## Telephony Phone Book Data Structures

This chapter describes the data structures used with the phone book service set of the telephony API.

### TelPhbEntryType

The TelPhbEntryType structure describes a single entry in a phone book.

```
typedef struct _TelPhbEntryType
{
    UInt16    phoneIndex;
    Char      *fullName;
    UInt8     fullNameSize;
    Char      *dialNumber;
```

## Telephony Phone Book

### Telephony Phone Book Data Structures

---

```
    UInt8    dialNumberSize;  
} TelPhbEntryType
```

#### Field Descriptions

- |                  |   |
|------------------|---|
| -> phoneIndex    | The index of the entry in the phone's phone book. This index is always zero-based.<br><br>The telephony manager is responsible for converting this index into the physical index in the phone, if required.   |
| <- fullName      | A buffer into which the retrieved full name of the entry is stored.<br><br>This string is stored using the local character set of the Palm <sup>TM</sup> handheld device. The telephony manager is responsible for converting the character set, if required.<br><br>Note that if this buffer is too small to contain the entire retrieved string, the string is truncated and the function using this structure generates a <code>telErrBufferSize</code> error.   |
| <-> fullNameSize | The size of the <code>fullName</code> string.<br><br>When the structure is used as an input parameter, this is the allocated size, in bytes, of the <code>fullName</code> buffer.<br><br>Upon return, this is the actual size of the string, including the null terminator character. If the <code>fullName</code> buffer is too small to contain all of the retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a <code>telErrBufferSize</code> error. |

<- dialNumber

A buffer into which the retrieved telephone number of the entry is stored.

Note that if this buffer is too small to contain the entire retrieved string, the string is truncated, and the function using this structure generates a `telErrBufferSize` error.

<-> dialNumberSize

The size of the `dialNumber` string.

When the structure is used as an input parameter, this is the allocated size, in bytes, of the `dialNumber` buffer.

Upon return, this is the actual size of the string, including the null terminator character. If the `dialNumber` buffer is too small to contain all of the retrieved string, this field is assigned the entire length of the string, and the function using this structure generates a `telErrBufferSize` error.

## TelPhbGetAvailablePhonebooksType

The [TelPhbGetAvailablePhonebooks](#) functions uses the `TelPhbGetAvailablePhonebooksType` structure to return a list of the phone books available on the phone.

```
typedef struct
    TelPhbGetAvailablePhonebooksType
{
    UInt16    count;
    UInt8    *phonebooksP;
} TelPhbGetAvailablePhonebooksType
```

### Field Descriptions

<-> count	The number of entries in the array referenced by phonebooksP.
	When the structure is used as an input parameter, this is the allocated size, in bytes, of the phonebooksP buffer. Upon return, this is the actual size of the retrieved data.
	Upon return, this is the actual number of phone book IDs that could be retrieved. If the phonebooksP buffer is too small to contain all of the IDs, this field is assigned the actual count, and the function using this structure generates a telErrBufferSize error.
<- phonebooksP	An array of retrieved phone book IDs. Each ID is one of the <a href="#">Phone Book Type Constants</a> .

### TelPhbGetEntriesType

The [TelPhbGetEntries](#) function uses the TelPhbGetEntriesType structure to return a list of phone entries.

```
typedef struct _TelPhbGetEntriesType
{
    UInt16           first;
    UInt16           count;
    TelPhbEntryType *entriesP;
} TelPhbGetEntriesType
```

### Field Descriptions

-> first	The index of the first entry in the array referenced by entriesP.
----------	---

<-> count

When the structure is used as an input parameter, this is the number of entries that you want retrieved.

Upon return, this is the actual number of entries that were retrieved.

<- entriesP

An array of pointers to retrieved [TelPhbEntryType](#) structures.

## TelPhbGetEntryCountType

The [TelPhbGetEntryCount](#) function uses the TelPhbGetEntryCountType structure to return information about the entries in the currently selected phone book.

```
typedef struct _TelPhbGetEntryCountType
{
    UInt16          slots;
    UInt16          count;
} TelPhbGetEntryCountType
```

### Field Descriptions

<- slots

The total number of entry slots available in the phone book.

c count

The number of filled slots in the phone book.

## TelPhbGetEntryMaxSizesType

The [TelPhbGetEntryMaxSizes](#) function uses the TelPhbGetEntryMaxSizesType structure to return size information about the entries in the currently selected phone book.

```
typedef struct _TelPhbGetEntryMaxSizeType
{
    UInt8          fullNameMaxSize;
    UInt8          dialNumberMaxSize;
} TelPhbGetEntryMaxSizesType
```

## **Telephony Phone Book**

### *Telephony Phone Book Constants*

---

#### **Field Descriptions**

- <- `fullNameMaxSize` The largest size of any `fullName` field in the phone book.
- <- `dialNumberMaxSize` The largest size of any `dialNumber` field in the phone book.

## **Telephony Phone Book Constants**

This section describes the constants used with the phone book service set of the telephony API.

### **Phone Book Type Constants**

The phone book type constants specify the type of phone book that is currently selected.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
<code>kTelPhbFixedPhonebook</code>	0	The phone book stored on the phone.
<code>kTelPhbSimPhonebook</code>	1	The phone book stored on the SIM card.
<code>kTelPhbPhonePhonebook</code>	2	The phone book stored on the phone.
<code>kTelPhbLastDialedPhonebook</code>	3	The phone book from which a telephone number was most recently dialed.
<code>kTelPhbSimAndPhonePhonebook</code>	4	The combined phone and SIM card phone books.

Constant	Value	Description
kTelPhbAdaptorPhonebook	5	The phone book stored on the telephone adaptor.
kTelPhbFirstOemPhonebook	6	The ID of the first OEM phone book.  You can specify additional OEM phone books by incrementing this value; for example, the second OEM phone book is specified as: kTelPhbFirstOemPhonebook +1

---

## Telephony Phone Book Functions

This section describes the functions used with the phone book service set of the telephony API.

### TelPhbAddEntry

**Purpose** Add or replace an entry in the currently selected phone book.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelPhbAddEntry(UINT16 iRefnum,  
 TelAppID iAppId, TelPhbEntryType *iEntryP,  
 UINT16 *ioTransIdP)`

<b>Parameters</b>	-> <code>iRefnum</code>	The telephony manager library reference number.
	-> <code>iAppId</code>	The telephone application attachment identifier for your application.
	-> <code>iEntryP</code>	A pointer to a <a href="#">TelPhbEntryType</a> structure that contains the new entry information.
	<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode      errNone upon success or an error code upon failure.

transId      The transaction ID of the operation.

paramP      Points to the [TelPhbEntryType](#) structure passed to this function in the iEntryP parameter.

functionId      kTelPhbAddEntryMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by iEntryP remains in memory until the asynchronous call completes.

---

**Comments** The phoneIndex field of the [TelPhbEntryType](#) structure referenced by iEntryP specifies the index at which to write the entry.

Before using this function, you should verify that it is available by calling the [TelIsPhbServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelPhbDeleteEntry](#), [TelPhbSelectPhonebook](#)

## TelPhbDeleteEntry

<b>Purpose</b>	Deletes an entry from the currently selected phone book.								
<b>Declared In</b>	TelephonyMgr.h								
<b>Prototype</b>	<pre>Err TelPhbDeleteEntry(UINT16 iRefnum, TelAppID iAppId, UINT16 iEntryIndex, UINT16 *ioTransIdP)</pre>								
<b>Parameters</b>	<table><tr><td>-&gt; iRefnum</td><td>The telephony manager library reference number.</td></tr><tr><td>-&gt; iAppId</td><td>The telephone application attachment identifier for your application.</td></tr><tr><td>-&gt; iEntryIndex</td><td>The zero-based, logical index of the entry that you want deleted. The Telephony Manager computes the physical index.</td></tr><tr><td>&lt;-&gt; ioTransIdP</td><td>Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.</td></tr></table>	-> iRefnum	The telephony manager library reference number.	-> iAppId	The telephone application attachment identifier for your application.	-> iEntryIndex	The zero-based, logical index of the entry that you want deleted. The Telephony Manager computes the physical index.	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
-> iRefnum	The telephony manager library reference number.								
-> iAppId	The telephone application attachment identifier for your application.								
-> iEntryIndex	The zero-based, logical index of the entry that you want deleted. The Telephony Manager computes the physical index.								
<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.								
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.								
<b>Asynchronous Result</b>	<p>The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:</p> <table><tr><td>returnCode</td><td>errNone upon success or an error code upon failure.</td></tr><tr><td>transId</td><td>The transaction ID of the operation.</td></tr><tr><td>paramP</td><td>Points to the unsigned integer value passed to this function in the iEntryIndex parameter.</td></tr><tr><td>functionId</td><td>kTelPhbDeleteEntryMessage</td></tr></table>	returnCode	errNone upon success or an error code upon failure.	transId	The transaction ID of the operation.	paramP	Points to the unsigned integer value passed to this function in the iEntryIndex parameter.	functionId	kTelPhbDeleteEntryMessage
returnCode	errNone upon success or an error code upon failure.								
transId	The transaction ID of the operation.								
paramP	Points to the unsigned integer value passed to this function in the iEntryIndex parameter.								
functionId	kTelPhbDeleteEntryMessage								

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `iEntryP` remains in memory until the asynchronous call completes.

---

- Comments** Before using this function, you should verify that it is available by calling the [TelIsPhbServiceAvailable](#) macro.
- Compatibility** Implemented only if [4.0 New Feature Set](#) is present.
- See Also** [TelPhbAddEntry](#), [TelPhbSelectPhonebook](#)

## TelPhbGetAvailablePhonebooks

- Purpose** Retrieve the list of all phone books available on the phone.
- Declared In** `TelephonyMgr.h`
- Prototype** `Err TelPhbGetAvailablePhonebooks (UInt16 iRefnum,  
TelAppID iAppId,  
TelPhbGetAvailablePhonebooksType *ioParamP,  
UInt16 *ioTransIdP)`
- Parameters**
- |                           |   |
|---------------------------|---|
| -> <code>iRefnum</code>   | The telephony manager library reference number.   |
| -> <code>iAppId</code>    | The telephone application attachment identifier for your application.   |
| <-> <code>ioParamP</code> | A pointer to a <a href="#">TelPhbGetAvailablePhonebooksType</a> structure that lists the available phone books.<br><br>On input, the count field of this structure specifies the allocated size of the <code>phonebookP</code> buffer. Upon return, the count field specifies the actual number of entries retrieved, even if they were truncated to fit into the buffer. |

<-> `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

`returnCode`      `errNone` upon success or an error code upon failure.

`transId`      The transaction ID of the operation.

`paramP`      Points to the [TelPhbGetAvailablePhonebooksType](#) structure passed to this function in the `iEntryP` parameter.

`functionId`      `kTelPhbGetAvailablePhonebooksMessage`

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` remains in memory until the asynchronous call completes.

---

**Comments** The phone book IDs are stored into the `phonebookP` field of the [TelPhbGetAvailablePhonebooksType](#) structure referenced by `ioParamP`. If the `phonebookP` buffer is too small to contain all of the IDs, the information is truncated and this function returns the `telErrBufferSize` error. The `count` field of the structure is always updated to contain the actual number of entries that were retrieved.

Before using this function, you should verify that it is available by calling the [TelIsPhbServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelPhbSelectPhonebook](#)

## TelPhbGetEntries

**Purpose** Retrieve a range of entries from the currently selected phone book.

**Declared In** `TelephonyMgr.h`

**Prototype**

```
Err TelPhbGetEntries(UInt16 iRefnum,
                      TelAppID iAppId, TelPhbGetEntriesType *ioParamP,
                      UInt16 *ioTransIdP)
```

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
<-> <code>ioParamP</code>	A pointer to a <a href="#">TelPhbGetEntriesType</a> structure that is updated with the phone book entry information. The first entry retrieved is specified in the <code>first</code> field of this structure, which is zero-based; the number of entries retrieved is specified by the <code>count</code> field. Thus, the last entry retrieved is specified by:

```
ioParamP->first +
           ioParamP->count - 1
```

Upon return, the `count` field of the structure is the number of entries that were actually retrieved.

The `entriesP` field of this structure is a buffer that you allocate to contain the required number of pointers. Each pointer references a [TelPhbEntryType](#) structure that you must also preallocate.

On input, the fullNameSize and dialNumberSize fields of this structure specify the allocated sizes of the fullName and dialNumber buffers. Upon return, the fullNameSize and dialNumberSize fields specify the actual sizes of the buffers, even if a string was truncated to fit into the buffer.

<-> `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

`resultCode`      `errNone` upon success or an error code upon failure

`transId`      The transaction ID of the operation.

`paramP`      Points to the [TelPhbGetEntriesType](#) structure passed to this function in the `ioEntriesP` parameter.

`functionId`      `kTelPhbGetEntriesMessage`

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by `ioParamP` and any structures that it references remain in memory until the asynchronous call completes.

---

**Comments** The phone book information is stored into the [TelPhbEntryType](#) structures that you preallocate and refer to in the `entriesP` field of

the [TelPhbGetEntriesType](#) referenced by the `ioParamP` parameter.

If any buffer in any of the `TelPhbEntryType` structures is too small, the string intended for that buffer is truncated, and this function returns the `telErrBufferSize` error. In any case, the `fullNameSize` and `dialNumberSize` fields of each `TelPhbEntryType` structure contain the actual size of their respective strings.

If any entries in the specified range are empty, the entry is not retrieved, and the `count` value in the structure is updated.

Before using this function, you should verify that it is available by calling the [TelIsPhbServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelPhbGetEntry](#), [TelPhbSelectPhonebook](#)

## TelPhbGetEntry

**Purpose** Retrieve one entry from the currently selected phone book.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelPhbGetEntry(UInt16 iRefnum,  
TelAppID iAppId, TelPhbEntryType *ioEntryP,  
UInt16 *ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
<-> <code>ioEntryP</code>	A pointer to a <a href="#">TelPhbEntryType</a> structure that is updated with the phone book entry information.

On input, the fullNameSize and dialNumberSize fields of this structure specify the allocated sizes of the fullName and dialNumber buffers. Upon return, the fullNameSize and dialNumberSize fields specify the actual sizes of the buffers, even if a string was truncated to fit into the buffer.

<-> `ioTransIdP` Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

`resultCode`      `errNone` upon success or an error code upon failure.

`transId`      The transaction ID of the operation.

`paramP`      Points to the [TelPhbEntryType](#) structure passed to this function in the `ioEntryP` parameter.

`functionId`      `kTelPhbGetEntryMessage`

**Comments** The phone book information is stored into the [TelPhbEntryType](#) that you preallocate. If either buffer in the structure is too small, the string intended for that buffer is truncated, and this function returns the `telErrBufferSize` error. In any case, the `fullNameSize` and `dialNumberSize` fields of the structure contain the actual size of their respective strings.

Before using this function, you should verify that it is available by calling the [TelIsPhbServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelPhbGetEntries](#), [TelPhbSelectPhonebook](#)

## TelPhbGetEntryCount

**Purpose** Retrieve the total number of entries, and the number of filled entries in the currently selected phone book.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelPhbGetEntryCount (UInt16 iRefnum,  
TelAppID iAppId,  
TelPhbGetEntryCountType *oParamP,  
UInt16 *ioTransIdP)`

**Parameters**

-> <code>iRefnum</code>	The telephony manager library reference number.
-> <code>iAppId</code>	The telephone application attachment identifier for your application.
<- <code>oParamP</code>	A pointer to a <a href="#">TelPhbGetEntryCountType</a> structure that is updated with information about the number of entries in the phone book.
<-> <code>ioTransIdP</code>	Set the value of this parameter to NULL to cause the function to execute synchronously.  If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns `errNone` if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode	errNone upon success or an error code upon failure.
transId	The transaction ID of the operation.
paramP	Points to the <a href="#">TelPhbGetEntryCountType</a> structure passed to this function in the oParamP parameter.
functionId	kTelPhbGetEntryCountMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by oParamP remains in memory until the asynchronous call completes.

---

<b>Comments</b>	The total number of slots and the number of filled slots in the currently selected phone book are stored in the <a href="#">TelPhbGetEntryCountType</a> structure referenced by oParamP. Before using this function, you should verify that it is available by calling the <a href="#">TelIsPhbServiceAvailable</a> macro.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">TelPhbSelectPhonebook</a>

## TelPhbGetEntryMaxSizes

<b>Purpose</b>	Retrieves the maximum buffer sizes of any entries in the currently selected phone book.					
<b>Declared In</b>	TelephonyMgr.h					
<b>Prototype</b>	<pre>Err TelPhbGetEntryMaxSizes(UInt16 iRefnum,                            TelAppID iAppId,                            TelPhbGetEntryMaxSizesType *oParamP,                            UInt16 *ioTransIdP)</pre>					
<b>Parameters</b>	-> iRefnum	The telephony manager library reference number.				
	-> iAppId	The telephone application attachment identifier for your application.				
	<- oParamP	A pointer to a <a href="#">TelPhbGetEntryMaxSizesType</a> structure that is updated with information about the maximum buffer sizes of entries in the phone book.				
	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.				
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.					
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:  <table><tr><td>returnCode</td><td>errNone upon success or an error code upon failure.</td></tr><tr><td>transId</td><td>The transaction ID of the operation.</td></tr></table>		returnCode	errNone upon success or an error code upon failure.	transId	The transaction ID of the operation.
returnCode	errNone upon success or an error code upon failure.					
transId	The transaction ID of the operation.					

paramP Points to the [TelPhbGetEntryMaxSizesType](#) structure passed to this function in the oParamP parameter.

functionId kTelPhbGetEntryMaxSizesMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the structure referenced by oParamP remains in memory until the asynchronous call completes.

---

**Comments** The maximum size of any full name entry and the maximum size of any telephone number entry in the currently selected phone book are stored in the [TelPhbGetEntryMaxSizesType](#) structure referenced by oParamP.

Before using this function, you should verify that it is available by calling the [TelIsPhbServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelPhbGetEntries](#), [TelPhbGetEntry](#)

## TelPhbGetSelectedPhonebook

**Purpose** Retrieve the ID of the currently selected phone book.

**Declared In** `TelephonyMgr.h`

**Prototype** `Err TelPhbGetSelectedPhonebook (UInt16 iRefnum,  
TelAppID iAppId, UInt8 *oPhbIdP,  
UInt16 *ioTransIdP)`

**Parameters** -> `iRefnum` The telephony manager library reference number.

-> `iAppId` The telephone application attachment identifier for your application.

## Telephony Phone Book

### Telephony Phone Book Functions

---

<- oPhbIdP A pointer to an unsigned byte value. Upon return, this is filled in with the identifier of the currently selected phone book. The identifier is one of the [Phone Book Type Constants](#).

<-> ioTransIdP Set the value of this parameter to NULL to cause the function to execute synchronously.

If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.

**Synchronous Result** Returns errNone if the function was successful or returns an error code if not successful.

**Asynchronous Result** The following fields are updated in the [TelEventType](#) event that is sent when the operation completes:

returnCode errNone upon success or an error code upon failure.

transId The transaction ID of the operation.

paramP Points to the unsigned integer value passed to this function in the oPhbIdP parameter.

functionId kTelPhbGetSelectedPhonebookMessage

---

**WARNING!** When using this function asynchronously, you must ensure that the value referenced by oPhbIdP remains in memory until the asynchronous call completes.

---

**Comments** Before using this function, you should verify that it is available by calling the [TelIsPhbServiceAvailable](#) macro.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [TelPhbSelectPhonebook](#)

## TelPhbSelectPhonebook

<b>Purpose</b>	Make the specified phone book the currently selected phone book.	
<b>Declared In</b>	TelephonyMgr.h	
<b>Prototype</b>	<pre>Err TelPhbSelectPhonebook(UInt16 iRefnum,                            TelAppID iAppId, UInt8 iPhbId,                            UInt16 *ioTransIdP)</pre>	
<b>Parameters</b>	- > iRefnum	The telephony manager library reference number.
	- > iAppId	The telephone application attachment identifier for your application.
	- > iPhbId	The identifier of the phone book that you want selected as the current phone book. This must be one of the <a href="#">Phone Book Type Constants</a> .
	<-> ioTransIdP	Set the value of this parameter to NULL to cause the function to execute synchronously. If this parameter is not NULL, the call executes asynchronously. Upon return from this function, this points to the transaction identifier associated with the asynchronous operation.
<b>Synchronous Result</b>	Returns errNone if the function was successful or returns an error code if not successful.	
<b>Asynchronous Result</b>	The following fields are updated in the <a href="#">TelEventType</a> event that is sent when the operation completes:	
	returnCode	errNone upon success or an error code upon failure.
	transId	The transaction ID of the operation.
	paramP	Points to the unsigned integer value passed to this function in the iPhbId parameter.
	functionId	kTelPhbSelectPhonebookMessage

## **Telephony Phone Book**

*Telephony Phone Book Functions*

---

- Comments** Before using this function, you should verify that it is available by calling the [TelIsPhbServiceAvailable](#) macro.
- Compatibility** Implemented only if [4.0 New Feature Set](#) is present.
- See Also** [TelPhbGetAvailablePhonebooks](#),  
[TelPhbGetSelectedPhonebook](#)

# **Part IV: Libraries**



# Internet Library

---

This chapter provides reference material for the Internet library API:

- [Internet Library Data Structures](#)
- [Internet Library Constants](#)
- [Internet Library Functions](#)

The header file `INetMgr.h` declares the Internet library API. For more information on the Internet library, see the chapter “[Network Communication](#)” in the *Palm OS Programmer’s Companion*, vol. II, *Communications*.

---

**NOTE:** The information in this chapter applies only to version 3.2 or later of the Palm OS® on Palm VII® devices. These features are implemented only if the [Wireless Internet Feature Set](#) is present.

---

---

**WARNING!** In future OS versions, PalmSource, Inc. does not intend to support or provide backward compatibility for the Internet library API documented in this chapter.

---

## Internet Library Data Structures

### **INetCompressionTypeEnum**

The `INetCompressionTypeEnum` enum indicates the type of compression used for data exchanged via a socket. One of these enumerated types is set as the value of the [inetSockSettingCompressionTypeID](#) socket setting (a read-only setting).

## Internet Library

### Internet Library Data Structures

---

```
typedef enum {
    inetCompressionTypeNone = 0,
    inetCompressionTypeBitPacked,
    inetCompressionTypeLZ77
} INetCompressionTypeEnum;
```

---

#### Value Descriptions

inetCompressionTypeNone	No compression.
inetCompressionTypeBitPacked	Custom 5-bit compression scheme. This is typically used for data sent from the Palm Web Clipping Proxy server.
inetCompressionTypeLZ77	Not used; reserved for future use.

## INetConfigNameType

The `INetConfigNameType` structure holds the name of an Internet library network **configuration**. A configuration is a set of specific values for the Internet library settings. The Internet library defines a set of built-in configuration aliases for common network setups. These aliases point to configurations instead of holding the actual values themselves. You can use an alias anywhere in the API you would use a configuration. System-defined configuration aliases are listed in “[Configuration Aliases](#)” on page 1850.

This structure is used in the functions

[`INetLibConfigIndexFromName`](#), [`INetLibConfigRename`](#), and [`INetLibConfigSaveAs`](#).

---

```
#define inetConfigNameSize 32;

typedef struct {
    Char name[inetConfigNameSize]; // name of configuration
} INetConfigNameType, *INetConfigNamePtr;
```

---

#### Field Description

name	A configuration name (up to 32 characters).
------	---

## **INetContentTypeEnum**

The INetContentTypeEnum enum specifies the type of content to be exchanged via a socket. One of these enumerated types is set as the value of the [inetSockSettingContentTypeID](#) socket setting (a read-only setting).

---

```
typedef enum {
    inetContentTypeTextPlain = 0,
    inetContentTypeTextHTML,
    inetContentTypeImageGIF,
    inetContentTypeImageJPEG,
    inetContentTypeApplicationCML,
    inetContentTypeImagePalmOS,
    inetContentTypeOther
} INetContentTypeEnum;
```

---

### **Value Descriptions**

inetContentTypeTextPlain	Not used
inetContentTypeTextHTML	Not used
inetContentTypeImageGIF	Not used
inetContentTypeImageJPEG	Not used
inetContentTypeApplicationCML	Compressed HTML content (format used by the Palm Web Clipping Proxy server and PQAs)
inetContentTypeImagePalmOS	Palm OS® bitmap
inetContentTypeOther	Some undefined content type

## **INetHTTPAttrEnum**

The INetHTTPAttrEnum enum specifies HTTP request and response attributes that are set by [INetLibSockHTTPAttrSet](#) and returned by [INetLibSockHTTPAttrGet](#).

## Internet Library

### Internet Library Data Structures

---

```
typedef enum {

    //-----
    // Request only attributes
    //-----
    // The following are ignored unless going through a CTP proxy
    inetHTTPAttrWhichPart, // (W) UInt32 (0 -> N)
    inetHTTPAttrIncHTTP, // (W) UInt32 (Boolean) only applicable
        // when inetHTTPAttrConvAlgorithm set to ctpConvNone
    inetHTTPAttrCheckMailHi, // (W) UInt32
    inetHTTPAttrCheckMailLo, // (W) UInt32
    inetHTTPAttrReqContentVersion, // (W) UInt32, desired content
        // version. Represented as 2 low bytes. Lowest
    byte is
        // minor version, next higher byte is major
    version.
    //-----
    // Response only attributes
    //-----
    //-
    // Server response info
    inetHTTPAttrRspSize, // (R) UInt32, entire HTTP Response size
        // including header and data
    inetHTTPAttrResult, // (R) UInt32 (ctpErrXXX when using CTP
    Proxy)
    inetHTTPAttrErrDetail, // (R) UInt32 (server/proxy err code
    when
        // using CTP Proxy)
    inetHTTPAttrReason, // (R) Char[]
    // Returned entity attributes
    inetHTTPAttrContentSize, // (R) UInt32
    inetHTTPAttrContentSizeUncompressed, // (R) UInt32 (in
    bytes)
    inetHTTPAttrContentSizeUntruncated, // (R) UInt32
    inetHTTPAttrContentSizeVersion, // (R) UInt32, actual content
    version.
        // Represented as 2 low bytes. Lowest byte is minor
        // version, next higher byte is major version.
    inetHTTPAttrContentCacheID, // (R) UInt32, cacheID for this
    item
    inetHTTPAttrReqSize // (R) UInt32 size of request sent
} INetHTTPAttrEnum;
```

---

### Value Descriptions

inetHTTPAttrWhichPart	An index to the part of the response data desired, if the response data is partitioned into chunks. Write-only.
inetHTTPAttrIncHTTP	A Boolean that, if set, causes HTTP header data to be included as part of the content when retrieving raw data. Applicable only when <code>inetSettingConvAlgorithm</code> is set to <code>ctpConvNone</code> . Write-only.
inetHTTPAttrCheckMailHi	High-order byte of ID for checking mail. Write-only.
inetHTTPAttrCheckMailLo	Low-order byte of ID for checking mail. Write-only.
inetHTTPAttrReqContentVersion	Desired content version. Represented as 2 low bytes. Lowest byte is minor version, next higher byte is major version. Write-only.
inetHTTPAttrRspSize	Size of entire HTTP (header and data). Read-only.
inetHTTPAttrResult	Transport protocol error code. Read-only.
inetHTTPAttrErrDetail	Server/proxy error code when using the Palm Web Clipping Proxy server. Read-only.
inetHTTPAttrReason	Transport protocol error message. Read-only.
inetHTTPAttrContentLength	Size of response data. Read-only.
inetHTTPAttrContentSizeUncompr essed	Size of uncompressed response data. Read-only.
inetHTTPAttrContentSizeUntrunc ated	Total size of response data (it may have been truncated to less than this). Read-only.

inetHTTPAttrContentVersion	Actual content version. Represented as 2 low bytes. Lowest byte is minor version, next higher byte is major version. Read-only.
inetHTTPAttrContentCacheID	Cache ID for this item. Read-only.
inetHTTPAttrReqSize	Size of request sent. Read-only.

## **INetSchemeEnum**

The INetSchemeEnum enum specifies a protocol (http, https, etc.) used by a socket. Specify one of these enumerated types for the INetSockSettingScheme socket setting and for the scheme parameter to the [INetLibSockOpen](#) call.

---

```
typedef enum {
    inetSchemeUnknown = -1,
    inetSchemeDefault = 0,    // not used

    inetSchemeHTTP, // http:
    inetSchemeHTTPS, // https:
    inetSchemeFTP, // ftp:
    inetSchemeGopher, // gopher:
    inetSchemeFile, // file:
    inetSchemeNews, // news:
    inetSchemeMailTo, // mailto:
    inetSchemePalm, // palm:
    inetSchemePalmCall, // palmcall:
    inetSchemeMail, // not applicable to URLs, but used
                    // for the INetLibSockOpen call when
                    // creating a socket for mail IO
    inetSchemeMac, // mac: - Mac file system HTML

    inetSchemeFirst = inetSchemeHTTP, // first one
    inetSchemeLast = inetSchemeMail // last one
} INetSchemeEnum;
```

---

### **Value Descriptions**

inetSchemeHTTP	Use the HTTP protocol.
inetSchemeHTTPS	Use the HTTPS protocol (for a secure connection).

inetSchemeFTP	Use the FTP protocol. Not implemented.
inetSchemeGopher	Use the Gopher protocol. Not implemented.
inetSchemeFile	Launch local PQA file
inetSchemeNews	Use the News protocol. Not implemented.
inetSchemeMailTo	Launch the local messaging application, passing a "to" address.
inetSchemePalm	Launches a local application database. The URL is expected to be in the form cccc.tttt, where cccc is a four character creator name and tttt is a four character database type. This pair of strings is used to identify an application database to receive the launch message via a call to <a href="#">SysUIAppSwitch</a> .
inetSchemePalmCall	Launches a local application database. The URL is expected to be in the form cccc.tttt, where cccc is a four character creator name and tttt is a four character database type. This pair of strings is used to identify an application database to receive the launch message via a call to <a href="#">SysAppLaunch</a> .
inetSchemeMail	Creates a socket for mail I/O.
inetSchemeMac	Handles opening Mac OS file system HTML URLs. For use by the Simulator only.

## INetSettingEnum

The INetSettingEnum enum specifies a setting to be returned or set by the [INetLibSettingGet](#) or [INetLibSettingSet](#) calls.

---

```
typedef enum {
    inetSettingCacheSize, // (RW) UInt32, max size of cache
    inetSettingCacheRef, // (R) DmOpenRef, ref of cache DB
    inetSettingNetLibConfig, // (RW) UInt32, NetLib config to use
    inetSettingRadioID, // (R) UInt32[2], the 64-bit radio ID
    inetSettingBaseStationID, // (R) UInt32, the radio base
    station Id
    inetSettingMaxRspSize, // (W) UInt32 (in bytes)
    inetSettingConvAlgorithm, // (W) UInt32 (CTPConvEnum)
    inetSettingContentWidth, // (W) UInt32 (in pixels)
```

## Internet Library

### Internet Library Data Structures

---

```
inetSettingContentVersion, // (W) UInt32, content version
(encoder
    // version)
inetSettingNoPersonalInfo, // (RW) UInt32, send no deviceID/
zipcode
inetSettingUserName,
inetSettingLast
} INetSettingEnum;
```

---

### Value Descriptions

inetSettingCacheSize	Maximum size of cache (in bytes).
inetSettingCacheRef	DmOpenRef, reference to cache database. Read-only.
inetSettingNetLibConfig	The index of the net library network configuration to use. This value is saved as part of the preferences for each Internet library configuration. A value of 0 means to use the current configuration.
inetSettingRadioID	64-bit radio ID. Read-only. Used for wireless connections only.
inetSettingBaseStationID	Radio base station ID. Read-only. Used for wireless connections only.
inetSettingMaxRspSize	Maximum response size, in bytes. The default is 1024 bytes. Write-only.
inetSettingConvAlgorithm	Content conversion desired. Write-only. Possible values include: ctpConvCML (use 5-bit compression scheme), ctpConvCML8Bit (use 5-bit compression scheme, but in 8-bit form for debugging), ctpConvCMLLZ77 (use LZ77 compression scheme), ctpConvNone (no conversion; data is returned in native format)
inetSettingContentWidth	Width of the display for content. The default setting is 160 (pixels). Write-only.

inetSettingContentVersion	Content version (encoder version). Write-only. This setting is used to let the server know what encoder version it should use to encode content sent to the Palm client. Normally you don't need to set this value as it is initialized by INetLibOpen. The default encoder version is 0x8001.
inetSettingNoPersonalInfo	Send no device ID or zipcode information to the proxy server. This value is saved as part of the preferences for each Internet library configuration.
inetSettingUserName	Not applicable.

## INetSockSettingEnum

The INetSockSettingEnum enum specifies a socket setting to be returned or set by the [INetLibSockSettingGet](#) or [INetLibSockSettingSet](#) calls.

---

```
typedef enum {
    inetSockSettingScheme, // (R) UInt32, INetSchemeEnum
    inetSockSettingSockContext, // (RW) UInt32,
    inetSockSettingCompressionType, // (R) Char[]
    inetSockSettingCompressionTypeID, // (R) UInt32
                                // (INetCompressionTypeEnum)
    inetSockSettingContentType, // (R) Char[]
    inetSockSettingContentTypeID, // (R) UInt32
                                (INetContentTypeEnum)
    inetSockSettingData, // (R) UInt32, pointer to data
    inetSockSettingDataHandle, // (R) UInt32, handle to data
    inetSockSettingDataOffset, // (R) UInt32, offset to data from
    handle
    inetSockSettingTitle, // (W) Char[]
    inetSockSettingURL, // (R) Char[]
    inetSockSettingIndexURL, // (RW) Char[]
    inetSockSettingFlags, // (RW) UInt16, one or more of
                        // inetOpenURLFlagXXX flags
    inetSockSettingReadTimeout, // (RW) UInt32, read timeout in
    ticks
    inetSockSettingContentVersion, // (R) UInt32, content version
    number
    inetSockSettingLast
} INetSockSettingEnum;
```

---

**Value Descriptions**

inetSockSettingScheme	Requested scheme; one of the <a href="#">INetSchemeEnum</a> values. Read-only.
inetSockSettingSockContext	Not used.
inetSockSettingCompressionType	Name of requested compression type. Read-only.
inetSockSettingCompressionTypeID	Requested compression type; one of the <a href="#">INetCompressionTypeEnum</a> values. Read-only.
inetSockSettingContentType	String containing the MIME type of the content. Used only on received raw data. Read-only.
inetSockSettingContentTypeID	Content type of socket data; one of the <a href="#">INetContentTypeEnum</a> values. Read-only.
inetSockSettingData	Pointer to socket data. Read-only.
inetSockSettingDataHandle	Handle to socket data. Read-only.
inetSockSettingDataOffset	Offset to socket data from handle. Read-only.
inetSockSettingTitle	Web page title. This value is written to the cache (and the Web Clipping Application Viewer uses it later in a history list of cache entries). Write-only.
inetSockSettingURL	URL of requested data. Read-only.
inetSockSettingIndexURL	Index (or master) URL of requested data (for cache indexing). This is the topmost web page in a group of hierarchical pages; it serves to group the pages together and to filter cache list results. The Web Clipping Application Viewer sets this to the URL of the active PQA, for all pages linked from the PQA.

inetSockSettingFlags	URL request flags; one or more of inetOpenURLFlag... flags (see <a href="#">URL Open Constants</a> ).
inetSockSettingReadTimeout	The default timeout value for reads when the application uses the event mechanism. The time since last receiving data from a socket is monitored and a timeout error status event is returned from INetLibGetEvent if the timeout is exceeded.
inetSockSettingContentVersion	Content version number. Read-only.

## INetStatusEnum

The INetStatusEnum enum specifies the status of the socket. The status is returned in the [inetSockStatusChangeEvent](#) event structure and by the call [INetLibSockStatus](#).

---

```
typedef enum {
    inetStatusNew, // just opened
    inetStatusResolvingName, // looking up host address
    inetStatusNameResolved, // found host address
    inetStatusConnecting, // connecting to host
    inetStatusConnected, // connected to host
    inetStatusSendingRequest, // sending request
    inetStatusWaitingForResponse, // waiting for response
    inetStatusReceivingResponse, // receiving response
    inetStatusResponseReceived, // response received
    inetStatusClosingConnection, // closing connection
    inetStatusClosed, // closed
    inetStatusAcquiringNetwork, // network temporarily
                           // unreachable; socket on
                           // hold
    inetStatusPrvInvalid = 30 // internal value, not returned by
                           // INetMgr. Should be
                           // last.
} INetStatusEnum;
```

---

## Internet Library

### *Internet Library Constants*

---

#### Value Descriptions

inetStatusNew	Just opened
inetStatusResolvingName	Looking up host address
inetStatusNameResolved	Found host address
inetStatusConnecting	Connecting to host
inetStatusConnected	Connected to host
inetStatusSendingRequest	Sending request
inetStatusWaitingForResponse	Waiting for response
inetStatusReceivingResponse	Receiving response
inetStatusResponseReceived	Response received
inetStatusClosingConnection	Closing connection
inetStatusClosed	Connection closed
inetStatusAcquiringNetwork	Network temporarily unreachable; socket on hold
inetStatusPrvInvalid	Not used

## Internet Library Constants

### Configuration Aliases

The constants listed here specify Internet library network configuration alias names. Most of the Internet library API requires a configuration index rather than a name. Use [INetLibConfigIndexFromName](#) to obtain the alias's index from the name. For more information, see [INetConfigNameType](#).

The following aliases are defined for configuration names:

<b>Alias</b>	<b>Name string</b>	<b>Description</b>
inetCfgNameDefault	".Default"	Initially points to a generic configuration with no proxy. This uses the configuration set by the user in the Network preferences panel.
inetCfgNameDefWireline	".DefWireline"	Initially points to a generic configuration with no proxy. This uses the configuration set by the user in the Network preferences panel.
inetCfgNameDefWireless	".DefWireless"	Initially points to a generic configuration with no proxy. This uses the configuration set by the user in the Network preferences panel.
inetCfgNameCTPDefault	".CTPDefault"	Initially points to either ".CTPWireless" (on Palm VII® units) or ".CTPWireline" (on all other units). On the Palm VII unit, the Web Clipping Application Viewer application uses this configuration.
inetCfgNameCTPWireline	".CTPWireline"	Initially points to a wireline configuration that uses the Palm Web Clipping Proxy server.
inetCfgNameCTPWireless	".CTPWireless"	Initially points to a wireless configuration that uses the Palm.Net™ wireless system and the Palm Web Clipping Proxy server.

## URL Info Constants

The `inetURLInfoFlag...` constants convey information about a URL and are returned by the function [INetLibURLGetInfo](#).

## Internet Library

### Internet Library Constants

---

Constant	Value	Description
inetURLInfoFlagIsSecure	0x0001	URL was encrypted.
inetURLInfoFlagIsRemote	0x0002	URL was retrieved from the net.
inetURLInfoFlagIsInCache	0x0004	URL is stored in the cache.

## URL Open Constants

The `inetOpenURLFlag...` constants control how the [INetLibURLOpen](#) call operates with respect to caching and encryption. These flags are also used for the [inetSockSettingFlags](#) socket setting.

Constant	Value	Description
inetOpenURLFlagLookInCache	0x0001	Read data from the cache, if available.
inetOpenURLFlagKeepInCache	0x0002	Store the item in the cache, overwriting any other entries with an equivalent URL.
inetOpenURLFlagForceEncOn	0x0008	Use encryption even if scheme does not desire it.
inetOpenURLFlagForceEncOff	0x0010	Do not use encryption even if scheme desires it.

# Internet Library Functions

## **INetLibCacheGetObject**

**Purpose** Returns information about an entry in the cache database, including a handle to the record. Either the URL or the unique ID can be used to find the cache entry.

**Declared In** INetMgr.h

**Prototype** Err INetLibCacheGetObject (UInt16 libRefnum,  
MemHandle clientParamH, UInt8 \*urlTextP,  
UInt32 uniqueID, INetCacheInfoPtr cacheInfoP)

**Parameters**

-> libRefnum	Refnum of the Internet library.
-> clientParamH	Inet handle allocated by <a href="#">INetLibOpen</a> .
-> urlTextP	Pointer to URL text string to find. If this parameter is NULL, then uniqueID is used to find the entry.
-> uniqueID	Unique ID of the cache entry to find. This value can be obtained by calling <a href="#">INetLibCacheList</a> . This parameter is ignored if urlTextP is specified.
<- INetCacheInfoPtr	Pointer to a structure where information about the cache entry is returned. See the Comments section for details.

### **Result**

0	No error
inetErrParamsInvalid	One or more of the parameters are invalid.

## Internet Library

### Internet Library Functions

---

**Comments** The `INetCacheInfoPtr` type returned from this function is defined as a pointer to an `INetCacheInfoType` structure, which has the following definition:

```
typedef struct {
    MemHandle recordH; // handle to the cache
    record
    INetContentTypeEnum contentType;
    INetCompressionTypeEnum encodingType;
    UInt32 uncompressedDataSize;
    UInt8 flags; // unused
    UInt8 reserved;
    UInt16 dataOffset; // offset to content
    UInt16 dataLength; // size of content
    UInt16 urlOffset; // offset to URL
    UInt32 viewTime; // time last viewed
    UInt32 createTime; // time entry was created
    UInt16 murlOffset; // offset to master URL
} INetCacheInfoType, *INetCacheInfoPtr;
```

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

## INetLibCacheList

**Purpose** Returns an item from the cache list, based on its URL and index in the list.

**Declared In** `INetMgr.h`

**Prototype** `Err INetLibCacheList (UInt16 libRefnum,  
MemHandle ineth, UInt8 *cacheIndexURLP,  
UInt16 *indexP, UInt32 *uidP,  
INetCacheEntryP cacheP)`

**Parameters** `-> libRefnum` Refnum of the Internet library.  
`-> ineth` Inet handle allocated by `INetLibOpen`.

-> cacheIndexURLP

Pointer to a master URL string. Cache entries indexed with this master URL are returned. The Web Clipping Application Viewer sets the master URL of a cache page to the URL of the active PQA, so all pages linked from the PQA have the same master URL.

<-> indexP

Pointer to the index of the entry. Specify an index to find entries at this index or higher in the list. Specify NULL to search from the beginning, the first time you call this function. The index of the entry following the one found is returned on exit.

<- uidP

Pointer to a long value where the unique ID of the found cache entry is returned.

<- cacheP

Pointer to a structure where information about the found cache entry is returned. See the Comments section for details.

## Result

0

No error

inetErrTypeNotCached

Cache entry under requested index not found

inetErrParamsInvalid

The cacheIndexURLP parameter is NULL

inetErrCacheInvalid

The cache database doesn't exist

## Comments

This function first sorts the list of cache entries by URL. Then it returns in uidP the unique ID of the first cache entry with an index equal to or greater than indexP. The indexP value is updated to point to the next entry upon return.

To generate a complete list of cache entries having the same master URL (as for a history list), call this function repeatedly, always specifying the updated index, until it returns the error `inetErrTypeNotCached`.

## Internet Library

### Internet Library Functions

---

Note that a URL can exist multiple times in the Web Clipping Application Viewer cache database, thus the need for the uidP value.

The INetCacheEntryP type returned from this function is defined as a pointer to an INetCacheEntryType structure, which has the following definition:

```
typedef struct {
    UInt8 *urlP; // ptr to URL string
    UInt16 urlLen; // length of URL string
    UInt8 *titleP; // ptr to title string
    UInt16 titleLen; // length of title string
    UInt32 lastViewed; // time last viewed
                    // seconds since 1/1/1904
    UInt32 firstViewed; // time first viewed
                    // seconds since 1/1/1904
} INetCacheEntryType, *INetCacheEntryP;
```

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

## INetLibCheckAntennaState

**Purpose** Checks the antenna state and displays a dialog asking the user to raise it if it is down.

**Declared In** INetMgr.h

**Prototype** Err INetLibCheckAntennaState (UInt16 refNum)

**Parameters** -> refNum Refnum of the Internet library.

### Result

0	The user raised the antenna.
---	------------------------------

netErrUserCancel	The user closed the dialog by tapping Cancel.
------------------	---

This call can also return data manager errors if it fails internally.

**Comments** Applications don't need to check the antenna state by using this call. If an application opens the Internet library, the Internet library checks the antenna state when needed and displays the dialog to prompt the user to raise the antenna.

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

## **INetLibClose**

**Purpose** Closes up and frees an inet handle. Closes or decrements the open count of the net library.

**Declared In** INetMgr.h

**Prototype** Err INetLibClose (UInt16 libRefnum,  
MemHandle ineth)

**Parameters** -> libRefnum Refnum of the Internet library.  
-> ineth Inet handle allocated by INetLibOpen.

**Result**

0 No error

**Comments** This call must be made by an application when it's done with the Internet library. It closes any Internet sockets open by the application, disposes the memory referenced by the given inet handle, and calls NetLibClose, if necessary, to close the net Library or decrement its open count.

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibOpen](#)

## **INetLibConfigAliasGet**

**Purpose** Determines to which configuration a built-in alias points.

**Declared In** INetMgr.h

**Prototype** Err INetLibConfigAliasGet (UInt16 refNum,  
                  UInt16 aliasIndex, UInt16 \*indexP,  
                  Boolean \*isAnotherAliasP)

**Parameters**

-> libRefnum	Refnum of the Internet library.
-> aliasIndex	Index of alias configuration to query. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by <a href="#">INetLibConfigList</a> , or the index returned by <a href="#">INetLibConfigIndexFromName</a> .
<- indexP	Pointer where the index of the configuration pointed to by aliasIndex is returned. 0 is returned if aliasIndex does not point to another configuration.
<- isAnotherAliasP	If *indexP is the index of another alias configuration, this Boolean is set to true.

### **Result**

0	No error
inetErrParamsInvalid	aliasIndex is not valid
inetErrConfigNotAlias	aliasIndex is not an alias configuration

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibConfigAliasSet](#)

## **INetLibConfigAliasSet**

**Purpose** Points any of the built-in aliases ("DefWireline", "DefWireless", etc.) to a given defined configuration.

**Declared In** INetMgr.h

**Prototype** Err INetLibConfigAliasSet (UInt16 refNum,  
UInt16 configIndex, UInt16 aliasToIndex)

**Parameters**

- > libRefnum Refnum of the Internet library.
- > configIndex Index of configuration to set. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by [INetLibConfigList](#), or the index returned by [INetLibConfigIndexFromName](#).
- > aliasToIndex Index of configuration that the alias identified by configIndex is to point to. Specify 0 to remove an existing alias assignment.

### **Result**

0	No error
inetErrConfigNotAlias	configIndex is not an alias configuration
inetErrParamsInvalid	configIndex or aliasToIndex is not valid
inetErrConfigCantPointToAlias	Alias doesn't point to a real entry

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibConfigAliasGet](#)

## Internet Library

### Internet Library Functions

---

## INetLibConfigDelete

**Purpose** Deletes a configuration.

**Declared In** INetMgr.h

**Prototype** Err INetLibConfigDelete (UInt16 refNum,  
                  UInt16 index)

**Parameters**

-> refnum	Refnum of the Internet library.
-> index	Index of configuration to delete. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by <a href="#">INetLibConfigList</a> , or the index returned by <a href="#">INetLibConfigIndexFromName</a> .

### Result

0	No error
inetErrParamsInvalid	Index not valid
inetErrConfigCantDelete	Attempted to delete an alias configuration

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibConfigIndexFromName](#), [INetLibConfigList](#)

## **INetLibConfigIndexFromName**

**Purpose** Returns the index of a configuration given it's name.

**Declared In** INetMgr.h

**Prototype** Err INetLibConfigIndexFromName (UInt16 refNum,  
INetConfigNamePtr nameP, UInt16 \*indexP)

**Parameters**

-> refnum	Refnum of the Internet library.
-> nameP	Pointer to an <a href="#">INetConfigNameType</a> structure that names the configuration whose index you want to get.
<- indexP	Pointer where the index of the configuration identified in nameP is returned.

### **Result**

0	No error
inetErrConfigNotFound	Could not find requested configuration name

**Comments** If you name an alias, this routine returns the index of the alias entry, not the configuration the alias points to. This way the alias can be pointed to a different configuration.

Applications should store the index of the configuration they're using, rather than the name, so that they won't be confused if the user edits the name of the configuration from the Preferences panel.

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibConfigList](#)

## Internet Library

### Internet Library Functions

---

## INetLibConfigList

**Purpose** Returns an array containing a list of the available Internet library network configurations.

**Declared In** INetMgr.h

**Prototype** Err INetLibConfigList (UInt16 refNum,  
INetConfigNameType nameArray[],  
UInt16 \*arrayEntriesP)

**Parameters**

-> refnum	Refnum of the Internet library.
-> nameArray	Pointer to an array of <a href="#">INetConfigNameType</a> structs that is to be filled in by this routine.
<-> arrayEntriesP	On entry, a pointer to the number of entries available in nameArray; on exit, a pointer to the total number of entries in the system (which could exceed the size of the array you pass in).

### Result

0	No error
---	----------

**Comments** This routine can be used to obtain a list of available configurations for selection by the user.

Note that the built-in alias configurations, which start with a period, should not be displayed to the user as selectable choices. They are designed for internal use by applications that need a predetermined type of service (like ".CTPWireless" for PQA applications.) Also, any configurations that start with an underscore, like "\_CTPRAM", should not be displayed. These typically are configurations created by the Internet library for internal use and cannot be edited using the Network preferences panel.

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibConfigMakeActive](#)

## **INetLibConfigMakeActive**

**Purpose** Makes the given configuration active without having to close and reopen the Internet library by using `INetLibOpen`.

**Declared In** `INetMgr.h`

**Prototype** `Err INetLibConfigMakeActive (UInt16 refNum,  
MemHandle ineth, UInt16 configIndex)`

**Parameters**

-> libRefnum	Refnum of the Internet library.
-> ineth	Inet handle allocated by <code>INetLibOpen</code> .
-> configIndex	Index of configuration to activate. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by <a href="#">INetLibConfigList</a> , or the index returned by <a href="#">INetLibConfigIndexFromName</a> .

### **Result**

0	No error
---	----------

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibConfigSaveAs](#), [INetLibConfigList](#),  
[INetLibConfigIndexFromName](#)

## **INetLibConfigRename**

**Purpose** Renames a configuration.

**Declared In** INetMgr.h

**Prototype** Err INetLibConfigRename (UInt16 refNum,  
                  UInt16 index, INetConfigNamePtr newNameP)

**Parameters**

-> libRefnum	Refnum of the Internet library.
-> index	Index of configuration to rename. This is the index of the configuration in the internal array of configurations stored by the system. This is the same as the index of the item in the array returned by <a href="#">INetLibConfigList</a> , or the index returned by <a href="#">INetLibConfigIndexFromName</a> .
-> newNameP	Pointer to an <a href="#">INetConfigNameType</a> structure holding the new name of the configuration. The name cannot start with a period or an underscore.

### **Result**

0	No error
inetErrConfigBadName	Trying to save as an alias (beginning with ".") or as a built-in configuration (beginning with "_").
inetErrParamsInvalid	Invalid index
inetErrConfigCantDelete	Configuration to be renamed is either an alias or a built-in configuration

**Comments** After renaming, the configuration index stays the same so that applications that are set up to use that configuration will still work

correctly. Note that built-in configuration aliases (ones that start with a period) cannot be renamed.

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

## INetLibConfigSaveAs

**Purpose** Saves the current network configuration settings under the given name.

**Declared In** INetMgr.h

**Prototype** Err INetLibConfigSaveAs (UInt16 refNum,  
MemHandle ineth, INetConfigNamePtr nameP)

**Parameters**

-> libRefnum	Refnum of the Internet library.
-> ineth	Inet handle allocated by <a href="#">INetLibOpen</a> .
-> nameP	Pointer to an <a href="#">INetConfigNameType</a> structure holding the name of the configuration. The name cannot start with a period or an underscore.

### Result

0	No error
inetErrConfigBadName	Trying to save as an alias (beginning with ".") or as a built-in configuration (beginning with "_").
inetErrConfigTooMany	The internal configurations table is full. No more entries can be stored.

**Comments** If the configuration name specified already exists, it is replaced with the new settings.

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

## INetLibGetEvent

**Purpose** A replacement for EvtGetEvent that informs an application of status changes to Internet sockets as well as user interface events.

**Declared In** INetMgr.h

**Prototype**

```
void INetLibGetEvent (UInt16 libRefnum,
                      MemHandle ineth, INetEventType *eventP,
                      Int32 timeout)
```

**Parameters**

-> libRefnum	Refnum of the Internet library.
-> ineth	Inet handle allocated by INetLibOpen, or NULL.
<-> eventP	The event structure is returned via this pointer.
-> timeout	Timeout in ticks. Specify evtWaitForever to wait forever.

### Result

0	No error
---	----------

**Comments** This call is designed to replace [EvtGetEvent](#) in applications which use the Internet library. For convenience, if ineth is NULL, INetLibGetEvent is equivalent to EvtGetEvent.

INetLibGetEvent returns two additional events besides those returned by EvtGetEvent: [inetSockReadyEvent](#) and [inetSockStatusChangeEvent](#).

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibSockStatus](#), [INetLibURLOpen](#), [INetLibSockOpen](#), [INetLibSockRead](#)

## **INetLibOpen**

**Purpose** Creates a new application inet handle structure. Opens or increments the open count of the net library.

**Declared In** INetMgr.h

**Prototype** Err INetLibOpen (UInt16 libRefnum, UInt16 config, UInt32 flags, DmOpenRef cacheRef, UInt32 cacheSize, MemHandle \*inetHP)

**Parameters**

-> libRefnum	Refnum of the Internet library. Pass the value "INet.lib" to <a href="#">SysLibFind</a> to return this refnum.
-> config	Indicates the type of network service desired by the application. Returned by <a href="#">INetLibConfigIndexFromName</a> .
-> flags	Currently unused; set to 0.
-> cacheRef	Document cache database reference. Obtain this by using one of the <a href="#">DmOpenDatabase...</a> calls. Pass NULL if you don't want to use a cache.
-> cacheSize	Maximum size of the document cache (in bytes). This is ignored if you pass NULL for cacheRef.
<- inetHP	Pointer to a handle variable.

### **Result**

0	No error
inetErrTooManyClients	Too many clients opened already
inetErrIncompatibleInterface	The net library is already open with an incompatible interface

## Internet Library

### Internet Library Functions

---

<b>Comments</b>	This call must be made by an application before it can use any other Internet library calls. This call opens the Internet library and returns a pointer to an inet handle, which is then passed to subsequent calls to the Internet library. Every application that opens the Internet library gets its own unique inet handle.  When an application is done using the Internet library, it must call <a href="#">INetLibClose</a> , which closes both the Internet library and the net library, if necessary.
<b>Compatibility</b>	Implemented only if <a href="#">Wireless Internet Feature Set</a> is present.
<b>See Also</b>	<a href="#">INetLibClose</a> , <a href="#">INetLibConfigIndexFromName</a>

## INetLibSettingGet

<b>Purpose</b>	Retrieves current settings for an inet handle.	
<b>Declared In</b>	<code>INetMgr.h</code>	
<b>Prototype</b>	<code>Err INetLibSettingGet (UInt16 libRefnum, MemHandle ineth, UInt16 setting, void *bufP, UInt16 *bufLenP)</code>	
<b>Parameters</b>	<code>-&gt; libRefnum</code>	Refnum of the Internet library.
	<code>-&gt; ineth</code>	Inet handle allocated by <a href="#">INetLibOpen</a> .
	<code>-&gt; setting</code>	The setting to get. Specify one of the <a href="#">INetSettingEnum</a> enumerated types.
	<code>&lt;- bufP</code>	Pointer to buffer where the return value is to be put.
	<code>&lt;-&gt; bufLenP</code>	Size of bufP on entry. Size of setting value on exit.

### Result

0	No error
---	----------

inetErrParamsInvalid	Invalid setting requested
inetErrSettingSizeInvalid	*bufLenP is the incorrect size for the requested setting

**Comments** This call can be used to retrieve the current settings of an inet handle. Some settings have default values that are stored in the system preferences database; see [INetSettingEnum](#) for details.

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibOpen](#), [INetLibSettingSet](#), [INetSettingEnum](#)

## INetLibSettingSet

**Purpose** Changes a setting for an inet handle.

**Declared In** INetMgr.h

**Prototype** Err INetLibSettingSet (UInt16 libRefnum,  
MemHandle ineth, UInt16 setting, void \*bufP,  
UInt16 bufLen)

**Parameters**

-> libRefnum	Refnum of the Internet library.
-> ineth	Inet handle allocated by <a href="#">INetLibOpen</a> .
-> setting	The setting to set. Specify one of the <a href="#">INetSettingEnum</a> enumerated types.
-> bufP	Pointer to the new setting value.
-> bufLen	Size of the value in bufP.

### Result

0	No error
---	----------

## Internet Library

### Internet Library Functions

---

inetErrParamsInvalid	Invalid setting specified
inetErrSettingSizeInvalid	bufLen is the incorrect size for the specified setting

**Comments** Any changes made to the settings last only as long as the `inetH` is around (until [INetLibClose](#) is called) and do not affect other applications that might be using the Internet library.  
An important note is that settings made through this call essentially change the default values for any sockets subsequently created through [INetLibURLOpen](#) or [INetLibSockOpen](#).

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibSettingGet](#), [INetSettingEnum](#)

## INetLibSockClose

**Purpose** Closes an inet socket handle.

**Declared In** INetMgr.h

**Prototype** Err INetLibSockClose (UInt16 libRefnum,  
MemHandle socketH)

**Parameters** -> libRefnum Refnum of the Internet library.  
-> socketH Handle of the socket to close.

### Result

0	No error
---	----------

**Comments** This call closes down and releases all memory associated with a socket created by [INetLibSockOpen](#) or [INetLibURLOpen](#).

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibOpen](#), [INetLibSockOpen](#), [INetLibURLOpen](#)

## INetLibSockConnect

**Purpose** Establishes a connection with a remote host.

**Declared In** INetMgr.h

**Prototype** Err INetLibSockConnect (UInt16 libRefnum,  
MemHandle sockH, UInt8 \*hostnameP, UInt16 port,  
Int32 timeout)

**Parameters**

-> libRefnum	Refnum of the Internet library.
-> sockH	Handle (allocated by <a href="#">INetLibSockOpen</a> or <a href="#">INetLibURLOpen</a> ) of the socket to connect.
-> hostnameP	Pointer to host name string; can be dotted decimal text string.
-> port	Port number, or 0 for default port.
-> timeout	Timeout in ticks. Specify <code>evtWaitForever</code> to wait forever.

### Result

0	No error
---	----------

**Comments** This call associates a remote host name and port number with a socket and, depending on the socket protocol, initiates a connection with that remote host.

This call may return immediately before actually finishing the connect. The application can simply go ahead and submit additional calls such as [INetLibSockRead](#), or it may wait for the connect to complete by either polling [INetLibSockStatus](#) until the socket status is `inetStatusConnected` (not recommended), or by waiting for an `inetSockStatusChangeEvent` event from [INetLibGetEvent](#) and checking the status then (preferred).

## Internet Library

### Internet Library Functions

---

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibSockOpen](#), [INetLibSockStatus](#), [INetLibGetEvent](#)

## INetLibSockHTTPAttrGet

**Purpose** Queries HTTP request header formed by the local host, or the response header information returned by a remote host.

**Declared In** INetMgr.h

**Prototype** Err INetLibSockHTTPAttrGet (UInt16 libRefnum,  
MemHandle sockH, UInt16 attr, UInt16 attrIndex,  
void \*bufP, UInt32 \*bufLenP)

**Parameters**

-> libRefnum	Refnum of the Internet library.
-> sockH	Handle (allocated by <a href="#">INetLibSockOpen</a> or <a href="#">INetLibURLOpen</a> ) of the socket.
-> attr	The attribute to get. Specify one of the <a href="#">INetHTTPAttrEnum</a> values.
-> attrIndex	The attribute index (if any). Currently unused.
<- bufP	Pointer to the address where the result is returned.
<-> bufLenP	Pointer to the size of bufP on entry; size of returned value on exit.

### Result

0	No error
inetErrSettingNotImplemented	Invalid setting specified
inetErrSettingSizeInvalid	bufLen is the incorrect size for the specified setting

<b>Comments</b>	This call queries either the request header formed by <code>INetLibSockHTTPReqCreate</code> and <code>INetLibSockHTTPAttrSet</code> , or the response header returned by the remote host.
<b>Compatibility</b>	Implemented only if <a href="#">Wireless Internet Feature Set</a> is present.
<b>See Also</b>	<a href="#">INetLibSockHTTPReqCreate</a>

## INetLibSockHTTPAttrSet

<b>Purpose</b>	Adds additional HTTP request headers to an HTTP request in a socket.														
<b>Declared In</b>	<code>INetMgr.h</code>														
<b>Prototype</b>	<pre>Err INetLibSockHTTPAttrSet (UInt16 libRefnum,                            MemHandle sockH, UInt16 attr, UInt16 attrIndex,                            UInt8 *bufP, UInt16 bufLen, UInt16 flags)</pre>														
<b>Parameters</b>	<table><tr><td>-&gt; libRefnum</td><td>Refnum of the Internet library.</td></tr><tr><td>-&gt; sockH</td><td>Handle (allocated by <code>INetLibSockOpen</code> or <code>INetLibURLOpen</code>) of the socket.</td></tr><tr><td>-&gt; attr</td><td>The attribute to set. Specify one of the <a href="#">INetHTTPAttrEnum</a> values.</td></tr><tr><td>-&gt; attrIndex</td><td>The attribute index (if any). Currently unused.</td></tr><tr><td>-&gt; bufP</td><td>Pointer to additional header text to add.</td></tr><tr><td>-&gt; bufLen</td><td>Length of bufP.</td></tr><tr><td>-&gt; flags</td><td>Flags that control the addition of new headers. Currently unused.</td></tr></table>	-> libRefnum	Refnum of the Internet library.	-> sockH	Handle (allocated by <code>INetLibSockOpen</code> or <code>INetLibURLOpen</code> ) of the socket.	-> attr	The attribute to set. Specify one of the <a href="#">INetHTTPAttrEnum</a> values.	-> attrIndex	The attribute index (if any). Currently unused.	-> bufP	Pointer to additional header text to add.	-> bufLen	Length of bufP.	-> flags	Flags that control the addition of new headers. Currently unused.
-> libRefnum	Refnum of the Internet library.														
-> sockH	Handle (allocated by <code>INetLibSockOpen</code> or <code>INetLibURLOpen</code> ) of the socket.														
-> attr	The attribute to set. Specify one of the <a href="#">INetHTTPAttrEnum</a> values.														
-> attrIndex	The attribute index (if any). Currently unused.														
-> bufP	Pointer to additional header text to add.														
-> bufLen	Length of bufP.														
-> flags	Flags that control the addition of new headers. Currently unused.														
<b>Result</b>	<table><tr><td>0</td><td>No error</td></tr></table>	0	No error												
0	No error														

## Internet Library

### Internet Library Functions

---

inetErrSettingNotImplemented	Invalid setting specified
inetErrSettingSizeInvalid	bufLen is the incorrect size for the specified setting

**Comments** This call modifies attributes of an HTTP request formed by `INetLibSockHTTPReqCreate`. Generally, attributes are set only before calling `INetLibSockHTTPReqSend`.

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibSockHTTPReqCreate](#), [INetLibSockHTTPReqSend](#)

## INetLibSockHTTPReqCreate

**Purpose** Forms an HTTP request for the socket.

**Declared In** `INetMgr.h`

**Prototype** `Err INetLibSockHTTPReqCreate (UInt16 libRefnum,  
MemHandle sockH, UInt8 *verbP, UInt8 *resNameP,  
UInt8 *refererP)`

**Parameters**

-> libRefnum	Refnum of the Internet library.
-> sockH	Handle (allocated by <code>INetLibSockOpen</code> or <code>INetLibURLOpen</code> ) of the socket.
-> verbP	Reserved for future use.
-> resNameP	Pointer to a string holding the name of the resource to get or put.
-> refererP	Pointer to a string holding the name of the referring URL, or <code>NULL</code> if none.

### Result

0	No error
---	----------

inetErrParamsInvalid	Not an HTTP socket
----------------------	--------------------

**Comments** This call forms an HTTP request for the socket. The request is not actually sent to the remote host until [INetLibSockHTTPReqSend](#) is called. The HTTP verb used in the request is determined by the value of the writeP parameter passed to [INetLibSockHTTPReqSend](#): if this parameter is NULL, "GET" is used. Otherwise, "POST" is used.

After a call to [INetLibSockHTTPReqCreate](#) but before a call to [INetLibSockHTTPReqSend](#), the application can add additional HTTP request headers using [INetLibSockHTTPAttrSet](#).

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibSockHTTPAttrSet](#), [INetLibSockHTTPReqSend](#)

## **INetLibSockHTTPReqSend**

**Purpose** Sends an HTTP request to the remote host or looks for data in the cache.

**Declared In** INetMgr.h

**Prototype** Err INetLibSockHTTPReqSend (UInt16 libRefnum,  
MemHandle sockH, void \*writeP, UInt32 writeLen,  
Int32 timeout)

**Parameters**

-> libRefnum	Refnum of the Internet library.
-> sockH	Handle (allocated by <a href="#">INetLibSockOpen</a> or <a href="#">INetLibURLOpen</a> ) of the socket.
-> writeP	Pointer to additional data to send after the request headers. Usually used for POST and PUT operations.
-> writeLen	Number of bytes in writeP.

## Internet Library

### Internet Library Functions

---

-> timeout      Timeout in ticks.

#### Result

0	No error
inetErrRequestTooLong	Request too big
inetErrEncryptionNotAvail	Encryption requested but not available

#### Comments

This call sends an HTTP request created by `INetLibSockHTTPReqCreate` and `INetLibSockHTTPAttrSet` to the remote host. If this is an POST or PUT operation, the data to write can be specified in `writeP`.

`INetLibSockHTTPReqSend` doesn't always do network I/O. If the proper socket flag is set, it checks first to see if the requested data is already in the cache. If it is, then a pointer to the cached data is stored in the socket and the socket status is updated to show that data is ready to be read. This will trigger an [inetSockReadyEvent](#) event.

The socket flag (`inetOpenURLFlagLookInCache`) that causes the cache to be checked first can be set via the `flags` parameter to [INetLibURLOpen](#) or by calling [INetLibSockSettingSet](#) with the appropriate setting (`inetSockSettingFlags`).

After sending the request, the application can wait for the response to arrive by either polling `INetLibSockStatus` until the `inputReady` boolean is set (not recommended), or by waiting for an `inetSockReadyEvent` event from `INetLibGetEvent` (preferred).

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibSockHTTPReqCreate](#), [INetLibSockHTTPAttrSet](#), [INetLibGetEvent](#)

## **INetLibSockOpen**

**Purpose** Creates and returns a new inet socket handle.

**Declared In** INetMgr.h

**Prototype** Err INetLibSockOpen (UInt16 libRefnum,  
MemHandle ineth, UInt16 scheme,  
MemHandle \*sockHP)

**Parameters**

-> libRefnum	Refnum of the Internet library.
-> ineth	Inet handle allocated by INetLibOpen.
-> scheme	The protocol scheme to use. Specify one of the <a href="#">INetSchemeEnum</a> types.
<- sockHP	Pointer to the address where the socket handle is returned.

### **Result**

0	No error
inetErrTooManySockets	Too many sockets open
inetErrSchemeNotSupported	Requested URL scheme not supported

**Comments** This call creates a new socket for the given scheme. No network I/O is performed. This is a relatively low level call that can be used in place of INetLibURLOpen when finer control over the socket settings is required.

Using INetLibURLOpen, an HTTP request can be handled with the simple sequence: INetLibURLOpen, INetLibSockRead, and INetLibSockClose. When using INetLibSockOpen, the same HTTP request would be handled by replacing the INetLibURLOpen call with the sequence: INetLibSockOpen, INetLibSockSettingSet (optional), INetLibSockConnect, INetLibSockHTTPReqCreate, INetLibSockHTTPAttrSet (optional), and INetLibSockHTTPReqSend.

## Internet Library

### Internet Library Functions

---

The use of `INetLibSockOpen` allows an application finer control over the socket settings (by calling `INetLibSockSettingSet`) and the HTTP request headers (by calling `INetLibSockHTTPAttrSet`).

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibOpen](#), [INetLibURLOpen](#), [INetLibSockRead](#),  
[INetLibSockClose](#), [INetLibSockSettingSet](#),  
[INetLibSockHTTPAttrSet](#)

## INetLibSockRead

**Purpose** Reads data from a socket.

**Declared In** `INetMgr.h`

**Prototype** `Err INetLibSockRead (UInt16 libRefnum,  
MemHandle sockH, void *bufP, UInt32 reqBytes,  
UInt32 *actBytesP, Int32 timeout)`

**Parameters**

-> libRefnum	Refnum of the Internet library.
-> sockH	Inet handle allocated by <code>INetLibOpen</code> .
-> bufP	Pointer to buffer where the data is placed.
-> reqBytes	Requested number of bytes.
<- actBytesP	Pointer to the actual number of bytes read.
-> timeout	Timeout in ticks. Specify <code>evtWaitForever</code> to wait forever.

### Result

0	No error
---	----------

**Comments** This call attempts to read `reqBytes` bytes from the given socket. It returns the actual number of bytes read in `*actBytesP`. If the connection with the remote host has been closed, `*actBytesP` contains 0 on exit.

Note that it is normal for the actual bytes read to be less than the requested number of bytes. The application should be prepared to call this routine repeatedly until the desired number of bytes have been read or until \*actBytesP contains 0, indicating the connection has been closed, or until an error is returned.

This call returns as much data as possible without blocking, however, if no data is available to be read, it does block until at least one byte is available.

Normally, applications will wait for an [inetSockReadyEvent](#) from [INetLibGetEvent](#) before calling INetLibSockRead.

Alternatively, the application could call [INetLibSockStatus](#) to determine if the socket has any data ready (not recommended), or could simply rely on INetLibSockRead to block until at least one byte is available to read. If no data is available before the timeout expires, `inetErrReadTimeout` error is returned.

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibURLOpen](#), [INetLibSockOpen](#), [INetLibSockStatus](#),  
[INetLibSockClose](#), [INetLibGetEvent](#)

## INetLibSockSettingGet

**Purpose** Retrieves a socket setting.

**Declared In** INetMgr.h

**Prototype** Err INetLibSockSettingGet (UInt16 libRefnum,  
MemHandle socketH, UInt16 setting, void \*bufP,  
UInt16 \*bufLenP)

**Parameters**

-> libRefnum	Refnum of the Internet library.
-> socketH	Handle (allocated by <a href="#">INetLibSockOpen</a> or <a href="#">INetLibURLOpen</a> ) of the socket to get a setting from.
-> setting	The setting to get. Specify one of the <a href="#">INetSockSettingEnum</a> values.

## Internet Library

### Internet Library Functions

---

<- bufP	Pointer to buffer where the setting value is to be placed.
<-> bufLenP	Size of bufP on entry; size of returned value on exit.

#### Result

0	No error
inetErrParamsInvalid	Invalid setting requested
inetErrSettingSizeInvalid	*bufLenP is the incorrect size for the requested setting

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibSockSettingSet](#)

## INetLibSockSettingSet

**Purpose** Changes a setting of a socket.

**Declared In** INetMgr.h

**Prototype** Err INetLibSockSettingSet (UInt16 libRefnum,  
MemHandle socketH, UInt16 setting, void \*bufP,  
UInt16 bufLen)

<b>Parameters</b>	-> libRefnum	Refnum of the Internet library.
	-> socketH	Handle (allocated by INetLibSockOpen or INetLibURLOpen) of the socket to set.
	-> setting	The setting to set. Specify one of the <a href="#">INetSockSettingEnum</a> values.
	-> bufP	Pointer to buffer containing the new setting value.

- > bufLen	Size of new setting in bufP.
<b>Result</b>	
0	No error
inetErrSettingNotImplemented	Invalid setting specified
inetErrSettingSizeInvalid	bufLen is the incorrect size for the setting

**Comments** This call can be use to override a general setting for a particular socket.

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibSockSettingGet](#)

## INetLibSockStatus

**Purpose** Retrieves the current status of a socket.

**Declared In** INetMgr.h

**Prototype** Err INetLibSockStatus (UInt16 libRefnum,  
MemHandle socketH, UInt16 \*statusP,  
Err \*sockErrP, Boolean \*inputReadyP,  
Boolean \*outputReadyP)

**Parameters**

- > libRefnum	Refnum of the Internet library.
- > socketH	Handle (allocated by <a href="#">INetLibSockOpen</a> or <a href="#">INetLibURLOpen</a> ) of the socket to get status on.
<- statusP	Pointer to the address where the status is returned. The status will be one of the <a href="#">INetStatusEnum</a> values.
<- sockErrP	Currently unused.

## Internet Library

### Internet Library Functions

---

<- inputReadyP Pointer to a Boolean; true is returned if the socket has data available to read.

<- outputReadyP  
Pointer to a Boolean; true is returned if the socket can accept data for writing.

#### Result

0	No error
---	----------

**Comments** Most applications that use `INetLibGetEvent` will rarely need to use this call since socket status changes are returned in the event structure.

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibURLOpen](#), [INetLibSockOpen](#), [INetLibSockRead](#), [INetLibGetEvent](#)

## INetLibURLCrack

**Purpose** Cracks a URL text string into its components.

**Declared In** `INetMgr.h`

**Prototype** `Err INetLibURLCrack (UInt16 libRefnum,  
                  UInt8 *urlTextP, INetURLType *urlP)`

**Parameters** `-> libRefnum` Refnum of the Internet library.  
`-> urlTextP` Pointer to URL text string.

<-> urlP      Pointer to address where the URL information block is to be returned.

### Result

0	No error
inetErrParamsInvalid	urlTextP is NULL or empty, or urlP is NULL
inetErrURLVersionInvalid	urlP is wrong version

### Comments

If a pointer member of urlP is set to NULL on entry, then on exit it will point to the start of that component within the original urlTextP string; the associated member length is set to the length of that URL component. If a pointer member of urlP is not NULL on entry, then it must point to a buffer of sufficient size to hold the member data, and on exit the component string will be copied into this buffer and the associated member length will be updated with the actual size. Note that the returned strings are not null-terminated, so the length values are important.

It's easiest to initialize the InetURLType block to zeros and let this function fill in all the information about the URL components.

The InetURLType block returned from this function has the following structure:

```
typedef struct {
    UInt16 version; // 0, for future compatibility
    UInt8 *schemeP; // ptr to scheme portion
    UInt16 schemeLen; // size of scheme portion
    UInt16 schemeEnum; // INetSchemeEnum; the
                       // scheme
    UInt8 *usernameP; // ptr to username portion
    UInt16 usernameLen; // size of username
    UInt8 *passwordP; // ptr to password portion
    UInt16 passwordLen; // size of password
    UInt8 *hostnameP; // ptr to host name portion
    UInt16 hostnameLen; // size of host name
    UInt16 port; // port number
    UInt8 *pathP; // ptr to path portion
    UInt16 pathLen; // size of path
```

## Internet Library

### Internet Library Functions

---

```
UInt8 *paramP; // ptr to param ( ;param)
UInt16 paramLen; // size of param
UInt8 *queryP; // ptr to query (?query)
UInt16 queryLen; // size of query
UInt8 *fragP; // ptr to fragment (#frag)
UInt16 fragLen; // size of fragment
} INetURLType
```

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

## INetLibURLGetInfo

**Purpose** Returns information about a URL.

**Declared In** INetMgr.h

**Prototype** Err INetLibURLGetInfo (UInt16 libRefnum,  
MemHandle ineth, UInt8 \*urlTextP,  
INetURLInfoType \*urlInfoP)

**Parameters**

-> libRefnum	Refnum of the Internet library.
-> ineth	Inet handle allocated by INetLibOpen.
-> urlTextP	Pointer to URL text string.
<-> urlInfoP	Pointer to address where the URL information structure is to be returned.

### Result

0	No error
inetErrParamsInvalid	urlInfoP is NULL or incorrect version

**Comments** The INetURLInfo block returned from this function has the following structure:

```
typedef struct {
UInt16 version; // 0, for future compatibility
UInt16 flags; // flags word
```

```
    UInt32 undefined; // reserved for future use  
} INetURLInfo
```

The flags word can consist of some combination of these values:

```
inetURLInfoFlagIsSecure // URL was encrypted  
inetURLInfoFlagIsRemote // URL was retrieved  
from the net  
inetURLInfoFlagIsInCache // URL is stored in  
the cache
```

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

## INetLibURLOpen

**Purpose** Accesses a URL on the Internet or in the cache.

**Declared In** INetMgr.h

**Prototype** Err INetLibURLOpen (UInt16 libRefnum,  
MemHandle ineth, UInt8 \*urlP,  
UInt8 \*cacheIndexURLP, MemHandle \*sockHP,  
Int32 timeout, UInt16 flags)

**Parameters**

-> libRefnum	Refnum of the Internet library.
-> ineth	Inet handle allocated by <a href="#">INetLibOpen</a> .
-> urlP	Pointer to string containing the URL to access.
-> cacheIndexURLP	Pointer to URL string under which the requested URL should be indexed in the cache. Specify NULL if you don't need to index the cache. If you are using the Web Clipping Application Viewer cache (not recommended), you must follow the Viewer convention, which is to pass the URL of the active PQA.
<- sockHP	Pointer to address where the socket handle is returned.

## Internet Library

### Internet Library Functions

---

-> timeout	Timeout in ticks. Specify evtWaitForever to wait forever.
-> flags	Flags indicating caching and encryption options desired. Specify zero, one, or more of the URL open flags (see <a href="#">URL Open Constants</a> ).

#### Result

0	No error
inetErrParamsInvalid	urlP is NULL

#### Comments

This call sets up a connection to a resource on the Internet addressed by urlP and returns a socket handle. Note that if you specify that the cache should be searched first, and if the data is found in the cache, no network I/O occurs. The application can then read that socket resource through the [INetLibSockRead](#) call.

This call is a convenience routine that internally makes the following calls for http URLs: [INetLibSockOpen](#), [INetLibSockConnect](#), [INetLibSockHTTPReqCreate](#), and [INetLibSockHTTPReqSend](#).

This routine returns immediately before performing any required network I/O. It is then up to the caller to either block on [INetLibSockRead](#), or to use [INetLibGetEvent](#) to model asynchronous operation. Using [INetLibGetEvent](#) is the preferred way of performing network I/O since it maximizes battery life and user-interface responsiveness.

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

**See Also** [INetLibSockOpen](#), [INetLibSockConnect](#), [INetLibSockRead](#), [INetLibSockClose](#)

## INetLibURLsAdd

**Purpose** Concatenates two URLs, resulting in one absolute URL.

**Declared In** INetMgr.h

<b>Prototype</b>	Err INetLibURLsAdd (UInt16 libRefnum, Char *baseURLStr, Char *embeddedURLStr, Char *resultURLStr, UInt16 *resultLenP)
<b>Parameters</b>	-> libRefnum Refnum of the Internet library. -> baseURLStr Pointer to base URL string. -> embeddedURLStr Pointer to URL text string to append. <-> resultURLStr Pointer to resulting URL string. <-> resultLenP Pointer to size of resultURLStr buffer on entry. On exit, pointer to resulting URL length (including null terminator).
<b>Result</b>	
	0 No error
<b>Comments</b>	Used to append a URL fragment to a base URL, resulting in an absolute URL string that can be passed to <a href="#">INetLibURLOpen</a> or other functions. This routine ensures that the resulting string conforms to the URL format.
<b>Compatibility</b>	Implemented only if <a href="#">Wireless Internet Feature Set</a> is present.

## Internet Library

### Internet Library Functions

---

## INetLibWiCmd

**Purpose** Invokes a command that operates on the wireless indicator.

**Declared In** INetMgr.h

**Prototype** Boolean INetLibWiCmd (UInt16 refNum, UInt16 cmd,  
int enableOrX, int y)

**Parameters**

-> refNum	Refnum of the Internet library.
-> cmd	The command to invoke. Specify one of the WiCmdEnum values (see Comments section).
-> enableOrX	If cmd is wiCmdSetEnabled, specify 1 to enable the wireless indicator or 0 to disable it. If cmd is wiCmdSetLocation, this specifies the x coordinate of the location.
-> y	The y coordinate of the location. Used only if cmd is wiCmdSetLocation.

**Result** If cmd is wiCmdEnabled, this function returns true if the wireless indicator is enabled or false if it is not. For other command types, the return value is undefined.

**Comments** The wireless indicator is a 19x13 pixel image on the screen to indicate the current wireless signal strength. This shows as 0 - 5 bars. If the application is in a non-modal window with a title bar, the preferred location for the indicator is at (140,1).

It automatically updates itself as long as you are calling INetLibGetEvent. It should be shown on screen while a wireless transaction is in progress. It may also be shown when the user has nothing useful to do next but initiate a wireless transaction, and there isn't much other useful information being displayed.

The WiCmdEnum enum specifies a command that operates on the wireless indicator in the user interface. The definition of this type is found in WirelessIndicator.h and is as follows:

```
typedef enum {  
    wiCmdInit = 0,
```

```
    wiCmdClear,  
    wiCmdSetEnabled,  
    wiCmdDraw,  
    wiCmdEnabled,  
    wiCmdSetLocation,  
    wiCmdErase  
} WiCmdEnum;
```

### **Value Descriptions**

wiCmdInit	Initializes the wireless indicator. You must invoke this command first, before using any of the others.
wiCmdClear	Applications shouldn't use this command. To erase the indicator, disable it by using wiCmdSetEnabled and passing 0 for enableOrX.
wiCmdSetEnabled	Enables or disables the wireless indicator.
wiCmdDraw	Redraws the wireless indicator using the latest data. Applications don't need to use this command since the indicator is redrawn automatically by INetLibGetEvent.
wiCmdEnabled	Returns a Boolean indicating if the wireless indicator is enabled.
wiCmdSetLocation	Sets the location for the wireless indicator on the screen.
wiCmdErase	Erases the wireless indicator. Applications shouldn't use this command. To erase the indicator, disable it by using wiCmdSetEnabled and passing 0 for enableOrX.

**Compatibility** Implemented only if [Wireless Internet Feature Set](#) is present.

## **Internet Library**

### *Internet Library Functions*

---

# PalmOSGlue Library

---

This chapter describes the API provided in the link library PalmOSGlue (`PalmOSGlue.lib` or `libPalmOSGlue.a`).

You use PalmOSGlue if you want to use the international and text manager features described in the chapter “[Localized Applications](#)” on page 363 in the *Palm OS Programmer’s Companion*, vol. I and you want to maintain backward compatibility with earlier releases of Palm OS®. If you link with PalmOSGlue, include the headers in the folder `Inc\Libraries\PalmOSGlue`, and make calls as they are listed in this chapter, then your code will run regardless of whether the device’s version of the operating system implements international support. The code in PalmOSGlue either uses the text manager or international manager on the ROM or, if the managers don’t exist, executes a simple Latin equivalent of the function.

---

**NOTE:** PalmOSGlue is a link library, not a shared library. Linking with this library increases your application’s code size. The amount by which your code size increases varies depending on the number of library functions you call. Use PalmOSGlue only on versions 2.0 and later of Palm OS.

---

In addition to covering the text and international manager API, PalmOSGlue adds some functions that are not included in any version of the Palm OS. This chapter describes the functions that are unique to PalmOSGlue and provides a mapping of PalmOSGlue calls to calls that exist in later versions of Palm OS.

## PalmOSGlue Functions

The following table shows the mapping between the functions declared in the glue headers and the international functions and macros. To learn more about a glue function, click the link in the right column.

This table lists only those functions that map to a function that exists in newer versions of the OS. The functions that are exclusive to PalmOSGlue are not listed. They are described following this table.

**Table 75.1 PalmOSGlue function mappings**

<b>This PalmOSGlue function...</b>	<b>...is identical to...</b>
BmpGlueGetBitDepth	<a href="#">BmpGetBitDepth</a>
BmpGlueGetDimensions	<a href="#">BmpGetDimensions</a>
BmpGlueGetNextBitmap	<a href="#">BmpGetNextBitmap</a>
DateGlueTemplateToAscii	<a href="#">DateTemplateToAscii</a>
DateGlueToDoWDMFormat	<a href="#">DateToDoWDMFormat</a>
FntGlueWCharWidth	<a href="#">FntWCharWidth</a>
FntGlueWidthToOffset	<a href="#">FntWidthToOffset</a>
LmGlueGetLocaleSetting	<a href="#">LmGetLocaleSetting</a>
LmGlueGetNumLocales	<a href="#">LmGetNumLocales</a>
LmGlueLocaleToIndex	<a href="#">LmLocaleToIndex</a>
LstGlueGetTopItem	<a href="#">LstGetTopItem</a>
OmGlueGetCurrentLocale	<a href="#">OmGetCurrentLocale</a>
OmGlueGetSystemLocale	<a href="#">OmGetSystemLocale</a>
ResGlueLoadConstant	<a href="#">ResLoadConstant</a>
SysGlueGetTrapAddress	<a href="#">SysGetTrapAddress</a>
TsmGlueGetFepMode	<a href="#">TsmGetFepMode</a>
TsmGlueSetFepMode	<a href="#">TsmSetFepMode</a>

**Table 75.1 PalmOSGlue function mappings (*continued*)**

<b>This PalmOSGlue function...</b>	<b>...is identical to...</b>
TblGlueGetNumberOfColumns	<a href="#"><u>TblGetNumberOfColumns</u></a>
TblGlueGetTopRow	<a href="#"><u>TblGetTopRow</u></a>
TblGlueSetSelection	<a href="#"><u>TblSetSelection</u></a>
TxtGlueByteAttr	<a href="#"><u>TxtByteAttr</u></a>
TxtGlueCaselessCompare	<a href="#"><u>TxtCaselessCompare</u></a>
TxtGlueCharAttr	<a href="#"><u>TxtCharAttr</u></a>
TxtGlueCharBounds	<a href="#"><u>TxtCharBounds</u></a>
TxtGlueCharEncoding	<a href="#"><u>TxtCharEncoding</u></a>
TxtGlueCharIsAlNum	<a href="#"><u>TxtCharIsAlNum</u></a>
TxtGlueCharIsAlpha	<a href="#"><u>TxtCharIsAlpha</u></a>
TxtGlueCharIsCntrl	<a href="#"><u>TxtCharIsCntrl</u></a>
TxtGlueCharIsDelim	<a href="#"><u>TxtCharIsDelim</u></a>
TxtGlueCharIsDigit	<a href="#"><u>TxtCharIsDigit</u></a>
TxtGlueCharIsGraph	<a href="#"><u>TxtCharIsGraph</u></a>
TxtGlueCharIsHex	<a href="#"><u>TxtCharIsHex</u></a>
TxtGlueCharIsLower	<a href="#"><u>TxtCharIsLower</u></a>
TxtGlueCharIsPrint	<a href="#"><u>TxtCharIsPrint</u></a>
TxtGlueCharIsPunct	<a href="#"><u>TxtCharIsPunct</u></a>
TxtGlueCharIsSpace	<a href="#"><u>TxtCharIsSpace</u></a>
TxtGlueCharIsUpper	<a href="#"><u>TxtCharIsUpper</u></a>
TxtGlueCharIsValid	<a href="#"><u>TxtCharIsValid</u></a>
TxtGlueCharSize	<a href="#"><u>TxtCharSize</u></a>
TxtGlueCharWidth	<a href="#"><u>FntWCharWidth</u></a>

## PalmOSGlue Library

### *PalmOSGlue Functions*

---

**Table 75.1 PalmOSGlue function mappings (*continued*)**

<b>This PalmOSGlue function...</b>	<b>...is identical to...</b>
<code>TxtGlueCharXAttr</code>	<a href="#"><u><code>TxtCharXAttr</code></u></a>
<code>TxtGlueCompare</code>	<a href="#"><u><code>TxtCompare</code></u></a>
<code>TxtGlueEncodingName</code>	<a href="#"><u><code>TxtEncodingName</code></u></a>
<code>TxtGlueGetChar</code>	<a href="#"><u><code>TxtGetChar</code></u></a>
<code>TxtGlueGetNextChar</code>	<a href="#"><u><code>TxtGetNextChar</code></u></a>
<code>TxtGlueGetPreviousChar</code>	<a href="#"><u><code>TxtGetPreviousChar</code></u></a>
<code>TxtGlueGetTruncationOffset</code>	<a href="#"><u><code>TxtGetTruncationOffset</code></u></a>
<code>TxtGlueMaxEncoding</code>	<a href="#"><u><code>TxtMaxEncoding</code></u></a>
<code>TxtGlueNextCharSize</code>	<a href="#"><u><code>TxtNextCharSize</code></u></a>
<code>TxtGlueParamString</code>	<a href="#"><u><code>TxtParamString</code></u></a>
<code>TxtGluePreviousCharSize</code>	<a href="#"><u><code>TxtPreviousCharSize</code></u></a>
<code>TxtGlueReplaceStr</code>	<a href="#"><u><code>TxtReplaceStr</code></u></a>
<code>TxtGlueSetNextChar</code>	<a href="#"><u><code>TxtSetNextChar</code></u></a>
<code>TxtGlueStrEncoding</code>	<a href="#"><u><code>TxtStrEncoding</code></u></a>
<code>TxtGlueTransliterate</code>	<a href="#"><u><code>TxtTransliterate</code></u></a>
<code>TxtGlueWordBounds</code>	<a href="#"><u><code>TxtWordBounds</code></u></a>
<code>WinGlueDrawChar</code>	<a href="#"><u><code>WinDrawChar</code></u></a>
<code>WinGlueDrawTruncChars</code>	<a href="#"><u><code>WinDrawTruncChars</code></u></a>

## BmpGlueGetCompressionType

**Purpose** Returns the compression type used for a specified bitmap.

**Declared In** BmpGlue.h

**Prototype** BitmapCompressionType BmpGlueGetCompressionType  
(const BitmapType \*bitmapP)

**Parameters** -> bitmapP Pointer to the bitmap.

**Result** Returns a BitmapCompressionType enum value. If bitmapP is NULL, or the specified bitmap is not compressed, this function returns BitmapCompressionTypeNone. If the specified bitmap's encoding version is less than 2, this function returns BitmapCompressionTypeScanLine.

**See Also** BmpCompress

## BmpGlueGetTransparentValue

**Purpose** Indicates if a specified bitmap has transparency—if, when the bitmap is drawn, pixels of a certain value won't be drawn.

**Declared In** BmpGlue.h

**Prototype** Boolean BmpGlueGetTransparentValue  
(const BitmapType \*bitmapP,  
 UInt32 \*transparentValueP)

**Parameters** -> bitmapP Pointer to the bitmap.

```
<- transparentValueP  
The pixel value that isn't drawn, if the bitmap  
has transparency. If the value returned by  
BmpGlueGetTransparentValue is false,  
*transparentValueP is left unchanged.
```

**Result** Returns true if, when drawing the specified bitmap, Palm OS doesn't draw pixels that have a value equal to that returned in transparentValueP. Returns false if all pixels are drawn.

**Comments** You can specify the transparent color when you create the bitmap using Constructor.

**See Also** [BmpGlueSetTransparentValue](#)

## BmpGlueSetTransparentValue

**Purpose** Causes pixels of a specified color not to be drawn when the bitmap is drawn.

**Declared In** BmpGlue.h

**Prototype** void BmpGlueSetTransparentValue  
(BitmapType \*bitmapP, UInt32 transparentValue)

**Parameters** -> bitmapP Pointer to the bitmap.  
-> transparentValueP  
The pixel value that isn't drawn.

**Result** Returns nothing.

**Comments** Does nothing if bitmapP is NULL, if bitmapP is an off-screen bitmap, or if transparentValue is invalid given the bitmap's bit depth.

If the specified bitmap's encoding version is less than 2, this function updates it to 2.

**See Also** [BmpGlueGetTransparentValue](#)

## CtlGlueGetControlStyle

<b>Purpose</b>	Return the type of the control, such as button, slider, and so on.
<b>Declared In</b>	<code>CtlGlue.h</code>
<b>Prototype</b>	<code>ControlStyleType CtlGlueGetControlStyle (const ControlType *ctlP)</code>
<b>Parameters</b>	<code>-&gt; ctlP</code> A pointer to a <a href="#">ControlType</a> structure.
<b>Result</b>	Returns one of the <a href="#">ControlStyleType</a> constants.
<b>Compatibility</b>	Implemented only in the PalmOSGlue library.

## CtlGlueGetFont

<b>Purpose</b>	Gets the font used when drawing a specified control's label.
<b>Declared In</b>	<code>CtlGlue.h</code>
<b>Prototype</b>	<code>FontID CtlGlueGetFont (const ControlType *ctlP)</code>
<b>Parameters</b>	<code>-&gt; ctlP</code> Pointer to the control object.
<b>Result</b>	Returns the ID of the font used to draw the control's label.
<b>See Also</b>	<a href="#">CtlGlueSetFont</a>

## CtlGlueGetGraphics

**Purpose** Gets the bitmaps displayed in place of a specified control's label.

**Declared In** CtlGlue.h

**Prototype** void CtlGlueGetGraphics (const ControlType \*ctlP,  
DmResID \*bitmapID, DmResID \*selectedBitmapID)

**Parameters**

-> ctlP	Pointer to the control.
<- bitmapID	Resource ID of the bitmap to display when the graphical control is not selected.
<- selectedBitmapID	Resource ID of the bitmap to display when the graphical control is selected, if, when selected, the graphical control should show a different bitmap.

**Result** Returns nothing.

**Comments** If the specified control is not a graphical control—one that displays a bitmap in place of the text label—\*bitmapID and \*selectedBitmapID are set to zero.

This function works with any graphical control, including sliders.

**Compatibility** Implemented only if 3.5 New Feature Set is present.

**See Also** [CtlGlueSetFont](#), CtlSetGraphics

## CtlGlueNewSliderControl

**Purpose** Create a new slider or feedback slider dynamically and install it in the specified form. The newly-created control is marked as a graphical control.

**Declared In** CtlGlue.h

**Prototype**

```
SliderControlType *CtlGlueNewSliderControl
(void **formPP, UInt16 ID,
ControlStyleType style, DmResID thumbID,
DmResID backgroundID, Coord x, Coord y,
Coord width, Coord height, UInt16 minValue,
UInt16 maxValue, UInt16 pageSize, UInt16 value)
```

**Parameters**

<-> formPP	Pointer to the pointer to the form in which the new control is installed. This value is not a handle; that is, the formPP value may change if the object moves in memory. In subsequent calls, always use the new formPP value returned by this function.
-> ID	Symbolic ID of the slider.
-> style	Either sliderCtl or feedbackSliderCtl. See ControlStyleType.
-> thumbID	Resource ID of the bitmap to display as the slider thumb. The slider thumb is the knob that the user can drag to change the slider's value. To use the default thumb bitmap, pass NULL for this parameter.
-> backgroundID	Resource ID of the bitmap to display as the slider background. To use the default background bitmap, pass NULL for this parameter.
-> x	Horizontal coordinate of the upper-left corner of the slider's boundaries, relative to the window in which it appears.

## PalmOSGlue Library

### *PalmOSGlue Functions*

---

-> y	Vertical coordinate of the upper-left corner of the slider's boundaries, relative to the window in which it appears.
-> width	Width of the slider, expressed in pixels. Valid values are 1–160.
-> height	Height of the slider, expressed in pixels. Valid values are 1–160.
-> minValue	Value of the slider when its thumb is all the way to the left.
-> maxValue	Value of the slider when its thumb is all the way to the right.
-> pageSize	Amount by which to increase or decrease the slider's value when the user clicks to the right or left of the thumb.
-> value	The initial value to display in the slider.

**Result** Returns a pointer to the new slider control. See *SliderControlType*.

**Compatibility** Implemented only if 3.5 New Feature Set is present.

**See Also** *CtlNewSliderControl*, *CtlNewGraphicControl*, *CtlNewControl*, *CtlValidatePointer*, *FrmRemoveObject*

## CtlGlueSetFont

**Purpose** Sets the font to use when drawing the control's label.

**Declared In** *CtlGlue.h*

**Prototype** *void CtlGlueSetFont (ControlType \*ctlP,  
FontID fontID)*

**Parameters** -> *ctlP* Pointer to the control object.

-> fontID      The ID of the font to use when drawing the control's label.

**Result**      Returns nothing.

**See Also**      [CtlGlueGetFont](#)

## CtlGlueSetLeftAnchor

**Purpose**      Causes a control's left bound to be fixed or to float.

**Declared In**      CtlGlue.h

**Prototype**      void CtlGlueSetLeftAnchor (ControlType \*ctlP,  
                         Boolean leftAnchor)

**Parameters**      -> ctlP      Pointer to the control.  
                         -> leftAnchor      A value of true causes the left bound of the control to be fixed.

**Result**      Returns nothing.

**Comments**      Used by controls that expand and shrink their width when the label is changed.

## FldGlueGetLineInfo

**Purpose**      Retrieve the word-wrapping information for a visible line within a field.

**Declared In**      FldGlue.h

**Prototype**      Boolean FldGlueGetLineInfo  
                          (const FieldType \*fldP, UInt16 lineNumber,  
                          UInt16 \*startP, UInt16 \*lengthP)

**Parameters**      -> fldP      A pointer to a [FieldType](#) structure.

## PalmOSGlue Library

### PalmOSGlue Functions

---

-> lineNumber	The number of the visible line in the field about which you want to retrieve information. Lines are numbered starting at 0.
<- startP	The byte offset into the <a href="#">FieldType</a> 's text field of the first character displayed by this line. If the line is blank, start is equal to the length of the field's text string.
<- lengthP	The length in bytes of the portion of the string displayed on this line. If the line is blank, the length is 0.

**Result** Returns `true` if `startP` and `endP` contain valid values, or `false` if the field is a single-line field or does not contain a line numbered `lineNum`.

**Compatibility** Implemented only in the PalmOSGlue library.

## FntGlueGetDefaultFontID

**Purpose** Return the font ID of a default font.

**Declared In** FntGlue.h

**Prototype** `FontID FntGlueGetDefaultFontID (`  
`FontDefaultType inFontType)`

**Parameters** `-> inFontType` A `FontDefaultType` constant specifying one of the system default fonts. This value can be one of the following:

- `defaultSystemFont`  
The default font for the system.
- `defaultLargeFont`  
The default large font.
- `defaultSmallFont`  
The default small font.

`defaultBoldFont`  
The default bold font.

**Result** Returns the ID of `inFontType`.

**Comments** Use this function whenever you need to obtain a font ID for one of the system default fonts. The default fonts (and thus, the IDs for the default fonts) vary depending on the system's locale. For example, Japanese systems have a different set of default fonts than systems using the Latin character encoding.

Use this function in place of the constants that specify the IDs of default fonts, as shown in the following table.

---

**In place of this...    ...use `FntGlueGetDefaultFontID` with this constant...**

---

<code>stdFont</code>	<code>defaultSystemFont</code> (best for displaying text) or: <code>defaultSmallFont</code> (if you want a smaller font)
<code>largeFont</code>	<code>defaultLargeFont</code>
<code>largeBoldFont</code>	<code>defaultLargeFont</code>
<code>boldFont</code>	<code>defaultBoldFont</code>

---

Note that `defaultSystemFont` and `defaultSmallFont` might return the same font ID or different font IDs, depending on the system locale.

**Compatibility** Implemented only in the PalmOSGlue library.

**See Also** [FontSelect](#), [FntSetFont](#), [FntSetFont](#)

## FrmGlueGetActiveField

**Purpose** Return the active field for a form.

**Declared In** FrmGlue.h

**Prototype** `extern FieldType *FrmGlueGetActiveField  
(const FormType *formP)`

**Parameters** `-> formP` Pointer to the form, or NULL to use the active form.

**Result** Returns a pointer to the active field. Returns NULL if there is no active form or field.

## FrmGlueGetDefaultButtonID

**Purpose** Gets the resource ID of the object on the form defined as the default button.

**Declared In** FrmGlue.h

**Prototype** `UInt16 FrmGlueGetDefaultButtonID  
(const FormType *formP)`

**Parameters** `-> formP` Pointer to the form.

**Result** Returns the resource ID of the object defined as the default button.

**See Also** [FrmDoDialog](#), [FrmGlueSetDefaultButtonID](#)

## FrmGlueGetHelpID

**Purpose** Gets the resource ID number of the form's help resource.

**Declared In** FrmGlue.h

**Prototype** UInt16 FrmGlueGetHelpID (const FormType \*formP)

**Parameters** -> formP Pointer to the form.

**Result** Returns the resource ID number of the form's help resource. The help resource is a String resource (type 'tSTR').

**See Also** [FrmGlueSetHelpID](#)

## FrmGlueGetLabelFont

**Purpose** Gets the font used for a particular label that appears on a form.

**Declared In** FrmGlue.h

**Prototype** FontID FrmGlueGetLabelFont  
(const FormType \*formP, UInt16 labelID)

**Parameters** -> formP Pointer to the form.

-> labelID ID of a label object in the form (the object's type must be frmLabelObj).

**Result** Returns a FontID value of 0 if either labelID is invalid or if the object indicated by labelID has a type other than frmLabelObj. Otherwise, this function returns the ID of the font used for the label.

**See Also** FrmGetObject Type, [FrmGlueSetLabelFont](#)

## FrmGlueGetMenuBarID

**Purpose** Gets the ID number of the form's menu bar.

**Declared In** FrmGlue.h

**Prototype** `UInt16 FrmGlueGetMenuBarID  
(const FormType *formP)`

**Parameters** `-> formP` Pointer to the form.

**Result** Returns the ID number of the form's menu bar, or zero if the form doesn't have a menu bar.

**See Also** [FrmSetMenu](#)

## FrmGlueGetObjectUsable

**Purpose** Returns whether an object in a form has been hidden.

**Declared In** FrmGlue.h

**Prototype** `Boolean  
FrmGlueGetObjectUsable (const FormType *formP,  
UInt16 objIndex)`

**Parameters** `-> formP` A pointer to a [FormType](#) structure.

`-> objIndex` The index of the object on the form.

**Result** Returns `true` if the object is usable, meaning that it is considered part of the user interface. Returns `false` if the object is not usable. Objects that are not usable never appear on the screen. The function [FrmHideObject](#) clears an object's usable bit to hide that the object.

**Comments** Implemented only in the PalmOSGlue library.

## FrmGlueSetDefaultButtonID

**Purpose** Designates the object on the form that is to act as the default button.

## Declared In FrmGlue.h

**Prototype** void FrmGlueSetDefaultButtonID (FormType \*formP,  
                  UInt16 defaultButton)

**Parameters**    -> formP              Pointer to the form.

-> defaultButton  
The resource ID of the object on the form that is to be the default button.

**Result** Returns nothing.

**See Also** [FrmDoDialog](#), [FrmGlueGetDefaultButtonID](#)

## FrmGlueSetHelpID

**Purpose** Designates the String resource that is to act as the form's help resource.

## Declared In FrmGlue.h

**Prototype** void FrmGlueSetHelpID (FormType \*formP,  
                  UInt16 helpRscID)

**Parameters**     $\rightarrow$  **formP**              Pointer to the form.

-> helpRscID The resource ID of the String resource that is to be the form's help resource.

**Result** Returns nothing.

#### **See Also**

## FrmGlueSetFontLabel

**Purpose** Sets the font used for a particular label that appears on a form.

**Declared In** FrmGlue.h

**Prototype** void FrmGlueSetFontLabel (FormType \*formP,  
                  UInt16 labelID, FontID fontID)

**Parameters**

-> formP	Pointer to the form.
-> labelID	ID of a label object in the form (the object's type must be frmLabelObj).
-> fontID	ID of the font to be used for the label.

**Result** Returns nothing.

**Comments** This function does nothing if either labelID is invalid or if the object indicated by labelID has a type other than frmLabelObj.

**See Also** FrmGetObjectType, [FrmGlueGetLabelFont](#)

## IntlGlueGetRoutineAddress

**Purpose** Return the address of a Text Manager function or of its PalmOSGlue equivalent.

**Declared In** IntlGlue.h

**Prototype** void \*IntlGlueGetRoutineAddress  
(IntlSelector selector, const void \*latinSymbol)

**Parameters**

-> selector	One of the routine selectors defined in IntlMgr.h.
-------------	--

-> latinSymbol    The corresponding `TxtLatinfunc` symbol defined in `IntlGlue.h`.

**Result**    Returns the address of the native Palm OS function if it is defined. If the function is not defined, it returns the address of the corresponding PalmOSGlue function.

**Comments**    Use `IntlGlueGetRoutineAddress` for performance reasons. You can use the address that it returns to call the function at that address without having to go through the International Manager's trap dispatch table. `IntlGlueGetRoutineAddress` is mostly useful for optimizing the performance of Text Manager functions that are called in a tight loop.

To call `IntlGlueGetRoutineAddress`, you must pass both the international trap for the function and the corresponding symbol. For example, to obtain the address of the `TxtGetNextChar` function or of its PalmOSGlue equivalent, you would make this call:

```
myTxtGetNextChar =  
    IntlGlueGetRoutineAddress(intlTxtGetNextChar,  
        TxtLatinGetNextChar);
```

The returned address is only valid while your application stays locked in memory. For this reason, you should only use the returned address up to the point where your application terminates. When the application starts up again, you should call `IntlGlueGetRoutineAddress` again.

**Compatibility**    Implemented only in the PalmOSGlue library.

**See Also**    [IntlGetRoutineAddress](#)

## LstGlueGetFont

**Purpose** Get the font used to draw a list's text strings.

**Declared In** LstGlue.h

**Prototype** FontID LstGlueGetFont (const ListType \*listP)

**Parameters** -> listP Pointer to the list.

**Result** Returns the ID of the font used to draw all list text strings.

**See Also** [LstGlueSetFont](#)

## LstGlueGetItemsText

**Purpose** Get the text strings that represent the items in a list.

**Declared In** LstGlue.h

**Prototype** Char \*\*LstGlueGetItemsText  
(const ListType \*listP)

**Parameters** -> listP Pointer to the list.

**Result** Returns a pointer to an array of pointers to the text of the list choices.

**See Also** [LstGetSelectionText](#), [LstSetListChoices](#)

## LstGlueSetFont

**Purpose** Specify the font to be used to draw a list's text strings.

**Declared In** LstGlue.h

**Prototype** void LstGlueSetFont (ListType \*listP,  
FontID fontID)

**Parameters** -> listP Pointer to the list.  
-> fontID ID of the font to be used to draw all list text  
strings.

**Result** Returns nothing.

**See Also** [LstGlueGetFont](#)

## LstGlueSetIncrementalSearch

**Purpose** Enables or disables incremental search for a sorted popup list.

**Declared In** LstGlue.h

**Prototype** void LstGlueSetIncrementalSearch  
(ListType \*listP, Boolean incrementalSearch)

**Parameters** -> listP Pointer to the list.  
-> incrementalSearch Set to true to enable incremental search,  
false to disable it.

**Result** Returns nothing.

**Comments** If incremental search is enabled, when the list is displayed the user can navigate the list by entering up to five characters. The list will scroll to present the first list item that matches the entered characters. This feature only works for popup lists, and only works

if the list is sorted and the list items are available to the List Manager (that is, you don't pass NULL to `LstSetListChoices`).

## SysGlueTrapExists

**Purpose** Macro that indicates if a given trap exists on the current system.

**Declared In** `SysGlue.h`

**Prototype** `SysGlueTrapExists (trapNum)`

**Parameters** `-> trapNum` One of the system trap constants.

**Result** Returns `true` if the current operating system defines the system trap `trapNum`, or `false` if the trap does not exist on that version of the operating system.

**Compatibility** Implemented only in the PalmOSGlue library.

## TblGlueGetColumnMasked

**Purpose** Determines whether a particular table column is masked.

**Declared In** `TblGlue.h`

**Prototype** `Boolean TblGlueGetColumnMasked  
(const TableType *tableP, Int16 column)`

**Parameters** `-> tableP` Pointer to the table.

`-> column` Column number (zero-based).

**Result** Returns `true` if the column is masked, `false` otherwise.

**Comments** If a table cell's column is masked and the cell's row is also masked, the table cell is drawn on the screen but is shaded to obscure the information that it contains.

**See Also** [TblSetColumnMasked](#)

## **TxtGlueCharIsVirtual**

**Purpose** Return whether a character is a virtual character or not.

**Declared In** [TxtGlue.h](#)

**Prototype** Boolean [TxtGlueCharIsVirtual](#) (UInt16 *inModifiers*, WChar *inChar*)

**Parameters** -> *inModifiers* The value passed in the *modifiers* field of the [keyDownEvent](#).

-> *inChar* A character.

**Result** Returns `true` if the character *inChar* is a virtual character, `false` otherwise.

**Comments** Virtual characters are nondisplayable characters that trigger special events in the operating system, such as displaying low battery warnings or displaying the keyboard dialog. Virtual characters should never occur in any data and should never appear on the screen.

Starting in Palm OS 3.1, the command modifier bit is always set in the *keyDownEvent* for a virtual character because the range for virtual characters overlaps the range for “real” characters that should appear on the screen. Earlier releases of the operating system did not always set the command modifier for virtual characters.

You can use this function to test whether a character is virtual or not. Pass the *chr* and *modifiers* fields exactly as you received them in the [keyDownEvent](#), and this function performs the appropriate check based on the operating system version.

**Compatibility** Implemented only in the PalmOSGlue library.

## **TxtGlueFindString**

**Purpose** Perform a case-insensitive search for a string in another string.

**Declared In** TextMgr.h

**Prototype** Boolean TxtGlueFindString  
(const Char \*inSourceStr,  
 const Char \*inTargetStr, UInt32 \*outPos,  
 UInt16 \*outLength)

**Parameters**

- > inSourceStr Pointer to the string to be searched.
- > inTargetStr Prepared version of the string to be found. This string should either be passed directly from the strToFind field in the [sysAppLaunchCmdFind](#) launch code's parameter block or it should be prepared using [TxtGluePrepFindString](#).
- <- outPos Pointer to the offset of the match in inSourceStr.
- <- outLength Pointer to the length in bytes of the matching text.

**Result** Returns true if the function finds inTargetStr within inSourceStr; false otherwise.

If found, the values pointed to by the outPos and outLength parameters are set to the starting offset and the length of the matching text. Unlike [TxtFindString](#), if the target string is not found, the values pointed to by outPos and outLength are not necessarily set to 0.

The search that TxtGlueFindString performs is locale-dependent. On most ROMs with Latin-based encodings, TxtGlueFindString returns true only if the string is at the beginning of a word. On Shift-JIS encoded ROMs,

`TxtGlueFindString` returns true if the string is located anywhere in the word.

<b>Comments</b>	Use this function instead of <a href="#">FindStrInStr</a> to support the global system find facility. This function contains an extra parameter, <code>outLength</code> , to specify the length of the text that matched. Pass this value to <a href="#">FindSaveMatch</a> in the <code>appCustom</code> parameter. Then when your application receives <code>sysAppLaunchCmdGoTo</code> , the <code>matchCustom</code> field contains the length of the matching text. You use the length of matching text to highlight the match within the selected record.  You must make sure that the parameters <code>inSourceStr</code> and <code>inTargetStr</code> point to the start of a valid character. That is, they must point to the first byte of a multi-byte character, or they must point to a single-byte character; if they don't, results are unpredictable.
<b>Compatibility</b>	Implemented only in the PalmOSGlue library.
<b>See Also</b>	<a href="#">TxtFindString</a> , <a href="#">TxtCaselessCompare</a>

## **TxtGlueGetHorizEllipsisChar**

<b>Purpose</b>	Return the horizontal ellipsis character.
<b>Declared In</b>	<code>TxtGlue.h</code>
<b>Prototype</b>	<code>WChar TxtGlueGetHorizEllipsisChar (void)</code>
<b>Parameters</b>	None.
<b>Result</b>	Returns the character code for horizontal ellipsis.
<b>Comments</b>	Versions 3.1 and higher of the Palm OS use different character codes for the horizontal ellipsis character and the numeric space character than earlier versions did. Use <code>TxtGlueGetHorizEllipsisChar</code> to return the appropriate code for horizontal ellipsis regardless of which version of Palm OS your application is run on.

**Compatibility** Implemented only in the PalmOSGlue library.

**See Also** [ChrHorizEllipsis](#), [TxtGlueGetNumericSpaceChar](#)

## **TxtGlueGetNumericSpaceChar**

**Purpose** Return the numeric space character.

**Declared In** `TxtGlue.h`

**Prototype** `WChar TxtGlueGetNumericSpaceChar (void)`

**Parameters** None.

**Result** Returns the character code for numeric space.

**Comments** Versions 3.1 and higher of the Palm OS use different character codes for the horizontal ellipsis character and the numeric space character than earlier versions did. Use `TxtGlueGetNumericSpaceChar` to return the appropriate code for numeric space regardless of which version of Palm OS your application is run on.

**Compatibility** Implemented only in the PalmOSGlue library.

**See Also** [ChrNumericSpace](#), [TxtGlueGetHorizEllipsisChar](#)

## **TxtGlueLowerChar**

**Purpose** Convert a character to lowercase.

**Declared In** `TxtGlue.h`

**Prototype** `WChar TxtGlueLowerChar (WChar inChar)`

**Parameters** `-> inChar` A character.

**Result** Returns the character as a lowercase letter.

<b>Comments</b>	This function does not handle the case in which the lowercase version of a character is represented by two or more characters. If you need to handle this situation, call the <a href="#">TxtGlueLowerStr</a> function instead of this one.
<b>Compatibility</b>	Implemented only in the PalmOSGlue library.
<b>See Also</b>	<a href="#">TxtGlueUpperChar</a> , <a href="#">TxtGlueLowerStr</a> , <a href="#">TxtGlueUpperStr</a> , <a href="#">TxtGlueTransliterate</a> , <a href="#">TxtTransliterate</a> , <a href="#">StrToLower</a>

## **TxtGlueLowerStr**

<b>Purpose</b>	Convert a string to all lowercase letters.
<b>Declared In</b>	<code>TxtGlue.h</code>
<b>Prototype</b>	<code>void TxtGlueLowerStr (Char* ioString, UInt16 inMaxLength)</code>
<b>Parameters</b>	 <code>&lt;-&gt; ioString</code> The string to be converted. <code>-&gt; inMaxLength</code> The size of the buffer that contains the string, excluding the terminating null character.
<b>Result</b>	Returns in <code>ioString</code> the input string with its letters converted to lowercase.
<b>Comments</b>	Converting a string from uppercase to lowercase letters or vice versa may change the size of the string. For this reason, you should always check the size of the <code>ioString</code> after this call returns.  You must make sure that the parameter <code>ioString</code> points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.  This function can only handle characters in the ISO Latin 1 character encoding unless the <a href="#">International Feature Set</a> is present.

**Compatibility** Implemented only in the PalmOSGlue library.

**See Also** [TxtGlueUpperStr](#), [TxtGlueLowerChar](#), [TxtGlueUpperChar](#),  
[StrToLower](#) [TxtGlueTransliterate](#), [TxtTransliterate](#)

## **TxtGluePrepFindString**

**Purpose** Set up for [TxtFindString](#) or [FindStrInStr](#).

**Declared In** `TxtGlue.h`

**Prototype** `void TxtGluePrepFindString (const Char* inSource,  
CharPtr outDest, UInt16 inDstSize)`

**Parameters**

-> inSource	Pointer to the text to be searched for. Must not be NULL.
<- outDest	The same text as in inSource but converted to a suitable format for searching. outDest must not be the same address as inSource.
-> inDstSize	The length in bytes of the area pointed to by outDest.

**Result** Returns in outDest an appropriately converted string.

**Comments** Use this function to normalize the string to search for before using [TxtGlueFindString](#), [TxtFindString](#), or [FindStrInStr](#) to perform a search that is internal to your application. If you use any of these three search routines in response to the [sysAppLaunchCmdFind](#) launch code, the string that the launch code passes in is already properly normalized for the search.

This function normalizes the string to be searched for. The method by which a search string is normalized varies depending on the version of Palm OS and the character encoding supported by the device.

Only inDstSize bytes of inSource are written to outDest. If necessary to prevent overflow of the destination buffer, not all of inSource is converted.

You must make sure that the parameter `inSource` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

<b>Compatibility</b>	Implemented only in the PalmOSGlue library.
----------------------	---

## **TxtGlueStripSpaces**

<b>Purpose</b>	Strip trailing and/or leading spaces from a string.
----------------	---

<b>Declared In</b>	<code>TxtGlue.h</code>
--------------------	------------------------

<b>Prototype</b>	<code>Char* TxtGlueStripSpaces (Char* ioStr, Boolean leading, Boolean trailing)</code>
------------------	--

<b>Parameters</b>	<code>&lt;-&gt; ioStr</code> Any string. <code>-&gt; leading</code> If true, strip the leading spaces from the string. <code>-&gt; trailing</code> If true, strip the trailing spaces from the string.
-------------------	--

<b>Result</b>	Returns <code>ioStr</code> with the specified spaces stripped from it. Note that this function both changes the <code>ioStr</code> buffer parameter and returns a pointer to it.
---------------	--

<b>Comments</b>	You must make sure that the parameter <code>ioStr</code> points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.
-----------------	--

<b>Compatibility</b>	Implemented only in the PalmOSGlue library.
----------------------	---

## **TxtGlueTruncateString**

<b>Purpose</b>	Determine if a string can be displayed in a given number of pixels. If not, truncate the string.
<b>Declared In</b>	<code>TxtGlue.h</code>
<b>Prototype</b>	<code>Boolean TxtGlueTruncateString (Char *ioStringP,                   UInt16 inMaxWidth)</code>
<b>Parameters</b>	<p><code>&lt;-&gt; ioStringP</code> A null-terminated string. Upon return, the string is truncated if necessary so that it can be displayed in <code>inMaxWidth</code> pixels.</p> <p><code>-&gt; inMaxWidth</code> The maximum width in pixels.</p>
<b>Result</b>	Returns <code>true</code> if the string was truncated, or <code>false</code> if the string can fit without truncation.
<b>Comments</b>	This function determines whether <code>ioStringP</code> can be displayed in the specified width without being truncated. If it can, <code>TxtGlueTruncateString</code> returns <code>false</code> . If the string must be truncated, this function truncates the string to one less than the number of characters that can fit in <code>inMaxWidth</code> and then appends an ellipsis (...) character. (If the boundary characters are narrower than the ellipsis, more than one character may be dropped to make room). If <code>inMaxWidth</code> is narrower than the width of an ellipsis, the string is set to the empty string.
<b>Compatibility</b>	Implemented only in the PalmOSGlue library.
<b>See Also</b>	<a href="#"><u>FntWidthToOffset</u></a> , <a href="#"><u>WinDrawTruncChars</u></a> , <a href="#"><u>TxtGetTruncationOffset</u></a>

## **TxtGlueUpperChar**

**Purpose** Convert a character to uppercase.

**Declared In** `TxtGlue.h`

**Prototype** `WChar TxtGlueUpperChar (WChar inChar)`

**Parameters** `-> inChar` Any character.

**Result** Returns the character as an uppercase letter.

**Comments** This function does not handle the case in which the uppercase version of a character is represented by two or more characters. If you need to handle this situation, call the [TxtGlueUpperStr](#) function instead of this one.

**Compatibility** Implemented only in the PalmOSGlue library.

**See Also** [TxtGlueLowerChar](#), [TxtGlueUpperStr](#) [TxtGlueLowerStr](#),  
[TxtGlueTransliterate](#), [TxtTransliterate](#) [StrToLower](#)

## **TxtGlueUpperStr**

**Purpose** Convert a string to all uppercase letters.

**Declared In** `TxtGlue.h`

**Prototype** `void TxtGlueUpperStr (Char* ioString,  
                  UInt16 inMaxLength)`

**Parameters** `<-> ioString` The string to be converted.

`-> inMaxLength` The size of the buffer that contains the string, excluding the terminating null character.

**Result** Returns in `ioString` the input string with its letters converted to uppercase.

**Comments** Converting a string from uppercase to lowercase letters or vice versa may change the size of the string. For this reason, you should always check the size of the `ioString` after this call returns.

You must make sure that the parameter `ioString` points to the start of a valid character. That is, it must point to the first byte of a multi-byte character or it must point to a single-byte character. If it doesn't, results are unpredictable.

This function can only handle characters in the ISO Latin 1 character encoding unless the [International Feature Set](#) is present.

**Compatibility** Implemented only in the PalmOSGlue library.

**See Also** [TxtGlueLowerStr](#), [TxtGlueUpperChar](#), [TxtGlueLowerChar](#),  
[TxtGlueTransliterate](#), [TxtTransliterate](#), [StrToLower](#)

## WinGlueGetFrameType

**Purpose** Gets the frame type for a specified window.

**Declared In** WinGlue.h

**Prototype** FrameType WinGlueGetFrameType  
(const WinHandle winH)

**Parameters** -> winH The window's handle.

**Result** Returns a FrameType value indicating the window's frame style.

**See Also** [WinGlueSetFrameType](#)

## WinGlueSetFrameType

**Purpose** Sets the type of frame to be used for a specified window.

**Declared In** WinGlue.h

**Prototype** void WinGlueSetFrameType (WinHandle winH,  
FrameType frame)

**Parameters** -> winH The window's handle.

-> frame The style of frame to be used.

**Result** Returns nothing.

**See Also** [WinGlueGetFrameType](#)

## PalmOSGlue Library

### *PalmOSGlue Functions*

---

# Bluetooth Library: General Functions

---

The Bluetooth library is a shared library that provides a direct interface to the Bluetooth communication capability of the Palm OS. This chapter presents reference material for the security and utility functions of the Bluetooth library API:

[Security Functions](#) Describes security functions that allow the application to manage a database of trusted devices. Trusted devices don't need to undergo authentication when they reconnect with the local device.

[Utility Functions](#) Describes utility functions and macros that are useful when working with the Bluetooth library. They perform such tasks as converting between host and network byte ordering, and converting between device addresses and strings.

The header file `BtLib.h` declares the Bluetooth library functions and macros. The header file `BtLibTypes.h` declares the data structures that you use with those functions and macros. For more information about using the Bluetooth library, see the *Palm OS Programmer's Companion Supplement: Bluetooth*.

## Security Functions

The Bluetooth security functions allow the application to manage a database of devices that have been bonded or paired. To understand this database, it is important to understand the difference between pairing and bonding.

Typically, devices must authenticate with each other every time they connect to each other. This process is called *pairing*. However, the

## Bluetooth Library: General Functions

### Security Functions

---

devices can *bond* instead. Bonding is similar to pairing except that both devices remember the link key for the connection. If the two devices ever want to connect to each other again, they don't need to repeat the pairing process.

Information about paired and bonded devices is stored in a device database on the local device. If a remote device has bonded with the local device, its record remains in the database until it is explicitly deleted. If the remote device has paired with the local device but not bonded with it, the record is removed when the connection to the remote device terminates.

Remote devices that have bonded with the local device are also called *trusted devices*.



## BtLibSecurityFindTrustedDeviceRecord

**Purpose** Search the device database for the device with the specified Bluetooth address. Return the index of the corresponding device record in the database.

**Declared In** BtLib.h

**Prototype** Err BtLibSecurityFindTrustedDeviceRecord  
(UInt16 btLibRefNum,  
BtLibDeviceAddressTypePtr addrP, UInt16 \*index)

**Parameters** -> btLibRefNum Reference number for the Bluetooth library.  
-> addrP Bluetooth address of remote device.  
<- index Index of the record.

**Result** Returns btLibErrNoError if successful. Returns btLibErrNotFound if a record with the specified remote device address could not be found.

**See Also** [BtLibSecurityRemoveTrustedDevice\\_Record](#),  
[BtLibSecurityGetTrustedDeviceRecordInfo](#)



## BtLibSecurityGetTrustedDeviceRecordInfo

**Purpose** Get information from a device record in the device database.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibSecurityGetTrustedDeviceRecordInfo
    (UInt16 btLibRefNum, UInt16 index,
     BtLibDeviceAddressTypePtr addrP,
     Char *nameBuffer, UInt8 *nameBufferSize,
     BtLibClassOfDeviceType *cod, Boolean *persistent)
```

**Parameters**

-> btLibRefNum	Reference number for the Bluetooth library.
-> index	Index of the record.
<- addrP	Bluetooth address of remote device.
<- nameBuffer	Pointer to buffer to store user-friendly name of remote device. You must allocate this buffer. Provide a NULL pointer if the user-friendly name is not needed.
<-> nameBufferSize	Size of the nameBuffer buffer on entry. On exit, the size of the name.
<- cod	Pointer to a <a href="#">BtLibClassOfDeviceType</a> representing the class of the device. You must allocate this structure. Provide a NULL pointer if the device class is not needed.
<- lastConnected	The date since the device last connected. This date is measured in seconds since midnight January 1, 1904. Provide a NULL pointer if the date of last connection is not needed.

## Bluetooth Library: General Functions

### Security Functions

---

<- persistent If true, the device is bonded and can connect to the local device without authentication. If false, the device is paired but not bonded—it will need to reauthenticate if it connects again. Provide a NULL pointer if this information is not needed.

**Result** Returns btLibErrNoError if successful. Returns dmErrIndexOutOfRange if a record with the specified index could not be found.

**See Also** [BtLibSecurityFindTrustedDeviceRecord](#)



## BtLibSecurityNumTrustedDeviceRecords

**Purpose** Return the number of bonded devices in the device database or return the total number of devices in the device database.

**Declared In** BtLib.h

**Prototype** UInt16 BtLibSecurityNumTrustedDeviceRecords  
(UInt16 btLibRefNum, Boolean persistentOnly)

**Parameters** -> btLibRefNum Reference number for the Bluetooth library.

-> persistentOnly  
true to obtain the total number of bonded devices in the database. These are the same devices that appear in the trusted devices list.  
false to obtain the total number of devices in the device database. This includes the devices that are bonded and the devices that are paired but not bonded.

**Result** Returns the requested number of device records.

**See Also** [BtLibSecurityFindTrustedDeviceRecord](#),  
[BtLibSecurityGetTrustedDeviceInfo](#)



**New**

## BtLibSecurityRemoveTrustedDeviceRecord

**Purpose** Remove a device record from the device database.

**Declared In** BtLib.h

**Prototype** Err BtLibSecurityRemoveTrustedDeviceRecord  
(UInt16 btLibRefNum, UInt16 index)

**Parameters** -> btLibRefNum Reference number for the Bluetooth library.  
-> index Index of the record to remove.

**Result** Returns btLibErrNoError if successful. Returns dmErrIndexOutOfRange if a record with the specified index could not be found.

**See Also** [BtLibSecurityFindTrustedDeviceRecord](#)

## Utility Functions

This section describes functions and macros that are useful when working with the Bluetooth library.



New

## BtLibAddrAToBtd

**Purpose** Convert an ASCII string a Bluetooth device address in colon-separated form to a 48-bit [BtLibDeviceAddressType](#).

**Declared In** BtLib.h

**Prototype**

```
Err BtLibAddrAToBtd (UInt16 btLibRefNum,
                      const Char *addressString,
                      BtLibDeviceAddressType *btDevP)
```

**Parameters**

-> btLibRefNum	Reference number for the Bluetooth library.
-> addressString	String containing ASCII colon-separated Bluetooth device address.
<- btDevP	Pointer to a <a href="#">BtLibDeviceAddressType</a> to store the converted device address.

**Result** Returns btLibErrNoError to indicate that the conversion was successful.

**See Also** [BtLibAddrBtdToA](#)



## New **BtLibAddrBtdToA**

**Purpose** Convert 48-bit [BtLibDeviceAddressType](#) to an ASCII string in colon-separated form.

**Declared In** BtLib.h

**Prototype** Err BtLibAddrBtdToA (UInt16 btLibRefNum,  
BtLibDeviceAddressType \*btDevP, Char \*spaceP,  
UInt16 spaceSize)

**Parameters**

- > btLibRefNum Reference number for the Bluetooth library.
- > btDevP Address of a Bluetooth device. This parameter must not be NULL.
- <- spaceP Pointer to a buffer to store the ASCII formatted Bluetooth devices address upon return. This parameter must not be NULL.
- > spaceSize Size of the spaceP buffer, in bytes. Must be at least 18.

**Result** Returns `btLibErrNoError` if successful. Returns `btLibErrParamErr` if

- `btDevP` is NULL
- `spaceP` is NULL
- `spaceSize` is less than 18, the number of bytes required to store the ASCII formatted address

**See Also** [BtLibAddrAToBtd](#)

## Bluetooth Library: General Functions

### Utility Functions

---



**New**

### BtLibL2CapHToNL

**Purpose** Macro that converts a 32-bit value from host to L2CAP byte order. L2CAP byte order is little endian.

**Declared In** BtLib.h

**Prototype** BtLibL2CapHToNL (value)

**Parameters** -> value      32-bit value to convert.

**Result** Returns value in L2CAP byte order.

**See Also** [BtLibL2CapHToNS](#), [BtLibL2CapNToHL](#), [BtLibL2CapNToHS](#)



**New**

### BtLibL2CapHToNS

**Purpose** Macro that converts a 16-bit value from host to L2CAP byte order. L2CAP byte order is little endian.

**Declared In** BtLib.h

**Prototype** BtLibL2CapHToNS (value)

**Parameters** -> value      16-bit value to convert.

**Result** Returns value in L2CAP byte order.

**See Also** [BtLibL2CapHToNL](#), [BtLibL2CapNToHS](#), [BtLibSdpNToHL](#)



## New **BtLibL2CapNToHL**

**Purpose** Macro that converts a 32-bit value from L2CAP to host byte order.  
L2CAP byte order is little endian.

**Declared In** BtLib.h

**Prototype** BtLibL2CapNToHL (value)

**Parameters** -> value      32-bit value to convert.

**Result** Returns value in host byte order.

**See Also** [BtLibL2CapNToHS](#), [BtLibL2CapHToNL](#), [BtLibL2CapHToNS](#)



## New **BtLibL2CapNToHS**

**Purpose** Macro that converts a 16-bit value from L2CAP to host byte order.  
L2CAP byte order is little endian.

**Declared In** BtLib.h

**Prototype** BtLibL2CapNToHS (value)

**Parameters** -> value      16-bit value to convert.

**Result** Returns value in host byte order.

**See Also** [BtLibL2CapNToHL](#), [BtLibL2CapHToNS](#), [BtLibL2CapHToNL](#)

## Bluetooth Library: General Functions

### Utility Functions

---



#### New **BtLibRfCommHToNL**

**Purpose** Macro that converts a 32-bit value from host to RFCOMM byte order. RFCOMM byte order is big endian.

**Declared In** BtLib.h

**Prototype** BtLibRfCommHToNL (value)

**Parameters** ->value        32-bit value to convertx

**Result** Returns value in RFCOMM byte order.

**See Also** [BtLibRfCommHToNS](#), [BtLibRfCommNToHL](#), [BtLibRfCommNToHS](#)



#### New **BtLibRfCommHToNS**

**Purpose** Macro that converts a 16-bit value from host to RFCOMM byte order. RFCOMM byte order is big endian.

**Declared In** BtLib.h

**Prototype** BtLibRfCommHToNS (value)

**Parameters** ->value        16-bit value to convertx

**Result** Returns value in RFCOMM byte order.

**See Also** [BtLibRfCommHToNL](#), [BtLibRfCommHToNS](#), [BtLibRfCommNToHL](#)



## New **BtLibRfCommNToHL**

**Purpose** Macro that converts a 32-bit value from RFCOMM to host byte order. RFCOMM byte order is big endian.

**Declared In** BtLib.h

**Prototype** BtLibRfCommNToHL (value)

**Parameters** ->value      32-bit value to convertx

**Result** Returns value in host byte order.

**See Also** [BtLibRfCommNToHS](#), [BtLibRfCommHToNL](#), [BtLibRfCommHToNS](#)



## New **BtLibRfCommNToHS**

**Purpose** Macro that converts a 16-bit value from RFCOMM to host byte order. RFCOMM byte order is big endian.

**Declared In** BtLib.h

**Prototype** BtLibRfCommNToHS (value)

**Parameters** ->value      16-bit value to convertx

**Result** Returns value in host byte order.

**See Also** [BtLibRfCommNToHL](#), [BtLibRfCommHToNS](#), [BtLibRfCommHToNL](#)

## Bluetooth Library: General Functions

### Utility Functions

---



New

#### BtLibSdpHToNL

**Purpose** Macro that converts a 32-bit value from host to Service Discovery Protocol (SDP) byte order. SDP byte order is big endian.

**Declared In** BtLib.h

**Prototype** BtLibSdpHToNL (value)

**Parameters** ->value      32-bit value to convertx

**Result** Returns value in SDP byte order.

**See Also** [BtLibSdpHToNS](#), [BtLibSdpNToHL](#), [BtLibSdpNToHS](#)



New

#### BtLibSdpHToNS

**Purpose** Macro that converts a 16-bit value from host to Service Discovery Protocol (SDP) byte order. SDP byte order is big endian.

**Declared In** BtLib.h

**Prototype** BtLibSdpHToNS (value)

**Parameters** ->value      16-bit value to convertx

**Result** Returns value in SDP byte order.

**See Also** [BtLibSdpHToNL](#), [BtLibSdpNToHS](#), [BtLibSdpNToHL](#)



## New **BtLibSdpNToHL**

**Purpose** Macro that converts a 32-bit value from Service Discovery Protocol (SDP) to host byte order. SDP byte order is big endian.

**Declared In** BtLib.h

**Prototype** BtLibSdpNToHL (value)

**Parameters** -> value      32-bit value to convertx

**Result** Returns value in host byte order.

**See Also** [BtLibSdpNToHS](#), [BtLibSdpHToNL](#), [BtLibSdpHToNS](#)



## New **BtLibSdpNToHS**

**Purpose** Macro that converts a 16-bit value from Service Discovery Protocol (SDP) to host byte order. SDP byte order is big endian.

**Declared In** BtLib.h

**Prototype** BtLibSdpNToHS (value)

**Parameters** ->value      16-bit value to convertx

**Result** Returns value in host byte order.

**See Also** [BtLibSdpNToHL](#), [BtLibSdpHToNS](#), [BtLibSdpHToNL](#)

## **Bluetooth Library: General Functions**

### *Utility Functions*

---

# Bluetooth Library: Management

---

The management API of the Bluetooth library supports the lower levels of the Bluetooth specification, specifically the radio, baseband, and Link Manager Protocol specifications. This chapter presents reference material for the management API:

## Bluetooth Management Data Structures

This section lists some of the more important types used by the Bluetooth library management functions.

## Management Callback Events

This section lists the management callback events. Most of the management functions are asynchronous. In other words, they start operations and return before the operations complete. To signal the application that management operations have completed, the Bluetooth library generates *management callback events* by calling a callback function.

## Management Event Status Codes

When a management event is generated, the status field of the associated [`BtLibManagementEventType`](#) provides information about why the event occurred. This section explains what these codes mean.

## Library Management Functions

This section describes the functions that open and close the shared library.

## **Bluetooth Library: Management**

### *Bluetooth Management Data Structures*

---

#### Management Functions

The management functions handle the lower levels of the Bluetooth specification, specifically the radio, baseband, and Link Manager Protocol specifications. These functions perform tasks that include discovering devices, working with Asynchronous Connectionless (ACL) links and piconets, and maintaining global settings for the Bluetooth library.

#### Application-Defined Functions

This section describes the callback functions that handle management events.

The header file `BtLib.h` declares the Bluetooth library functions and macros. The header file `BtLibTypes.h` declares the data structures that you use with those functions and macros. For more information about using the Bluetooth library, see the *Palm OS Programmer's Companion Supplement: Bluetooth*.

## **Bluetooth Management Data Structures**

This section lists some of the more important types used by the Bluetooth library management functions.



### **BtLibAccessibleModeEnum**

---

The `BtLibAccessibleModeEnum` enum specifies a device's accessibility modes. See the "Generic Access Profile" chapter of the *Specification of the Bluetooth System* for more information about accessibility.

```
typedef enum {
    btLibNotAccessible = 0x00,
    btLibConnectableOnly = 0x02,
    btLibDiscoverableAndConnectable = 0x03
} BtLibAccessibleModeEnum;
```

### Value Descriptions

`btLibConnectableOnly`

The device responds to a page but not an inquiry.

`btLibDiscoverableAndConnectable`

The device responds to both a page and an inquiry.

`btLibNotAccessible`

The device does not respond to a page or an inquiry.



**New**

## BtLibClassOfDeviceType

The `BtLibClassOfDeviceType` type represents the class of the device and the services it supports.

```
typedef UInt32 BtLibClassOfDeviceType;
```

A device can support multiple services but only belongs to a single class. The class is specified in two parts: the major class, which broadly classifies the type of device, and the minor class, which together with the major class specifies the type of device in more detail.

An example is a simple cellular telephone. It provides Telephony services. Its major device class is Phone, and its minor device class is Cellular.

The *Bluetooth Assigned Numbers* specification defines a device class as having three bit fields. One field specifies the major service classes supported by the device. Another field specifies the major device class. The third field specifies the minor device class.

The constants provided here allow you to construct a device class that conforms to the Bluetooth specification. You simply perform a logical OR of the constants representing the service classes the device supports, the constant representing the device's major class, and the constant representing the device's minor class.

For example, device class of the simple cellular telephone can be computed as follows:

## **Bluetooth Library: Management**

### *Bluetooth Management Data Structures*

---

```
cellPhoneCOD = btLibCOD_Telephony |  
    btLibCOD_Major_Phone |  
    BtLibCOD_Minor_Phone_Cellular;
```

Constants are also provided to mask the individual bit fields in a device class.

### **Major Service Classes**

These constants define the Bluetooth major service classes. The service classes are described in the *Specification of the Bluetooth System*.

```
btLibCOD_LimitedDiscoverableMode  
btLibCOD_Networking  
btLibCOD_Rendering  
btLibCOD_Capturing  
btLibCOD_ObjectTransfer  
btLibCOD_Audio  
btLibCOD_Telephony  
btLibCOD_Information
```

### **Major Device Classes**

These constants define the Bluetooth major device classes. The major device classes are described in the *Specification of the Bluetooth System*.

```
btLibCOD_Major_Misc  
btLibCOD_Major_Computer  
btLibCOD_Major_Phone  
btLibCOD_Major_Lan_Access_Point  
btLibCOD_Major_Audio  
btLibCOD_Major_Peripheral  
btLibCOD_Major_Unclassified
```

### **Computer Minor Device Classes**

These constants define the minor device classes associated with the computer major class. They are described in the *Bluetooth Assigned Numbers* specification.

```
btLibCOD_Minor_Comp_Unclassified  
btLibCOD_Minor_Comp_Desktop  
btLibCOD_Minor_Comp_Server
```

btLibCOD\_Minor\_Comp\_Laptop  
btLibCOD\_Minor\_Comp\_Handheld  
btLibCOD\_Minor\_Comp\_Palm

### **Phone Minor Device Classes**

These constants define the minor device classes are associated with the computer major class. They are described in the *Bluetooth Assigned Numbers* specification.

btLibCOD\_Minor\_Phone\_Unclassified  
btLibCOD\_Minor\_Phone\_Cellular  
btLibCOD\_Minor\_Phone\_Cordless  
btLibCOD\_Minor\_Phone\_Smart  
btLibCOD\_Minor\_Phone\_Modem

### **LAN Access Point Minor Device Classes**

These constants define load factors for the LAN access point major device class. LAN access point load factors are described in more detail in the *Bluetooth Assigned Numbers* specification.

btLibCOD\_Minor\_Lan\_0  
    Fully available  
  
btLibCOD\_Minor\_Lan\_17  
    1-17% utilized  
  
btLibCOD\_Minor\_Lan\_33  
    17-33% utilized  
  
btLibCOD\_Minor\_Lan\_50  
    33-50% utilized  
  
btLibCOD\_Minor\_Lan\_67  
    50-67% utilized  
  
btLibCOD\_Minor\_Lan\_83  
    67-83% utilized  
  
btLibCOD\_Minor\_Lan\_99  
    83-99% utilized  
  
btLibCOD\_Minor\_Lan\_NoService  
    Fully utilized

## Bluetooth Library: Management

### Bluetooth Management Data Structures

---

#### Audio Minor Device Classes

These constants define the minor classes associated with the audio major class. They are described in more detail in the *Bluetooth Assigned Numbers* specification.

`btLibCOD_Minor_Audio_Unclassified`  
`btLibCOD_Minor_Audio_Headset`

#### Masks

These constants define bit masks to isolate certain fields of the device class.

`btLibCOD_Service_Mask`  
A mask to isolate the major service class field from the other fields of the device class.

`btLibCOD_Major_Mask`  
A mask to isolate the major device class field from the other fields of the device class.

`btLibCOD_Minor_Mask`  
A mask to isolate the minor device class field from the other fields of the device class.

`btLibCOD_ServiceAny`  
Used as a device filter for the [BtLibDiscoverMultipleDevices](#) and [BtLibDiscoverSingleDevice](#) functions. With this filter, devices providing any service appear in the device list. Same as `btLibCOD_Service_Mask`.

`btLibCOD_Major_Any`  
Used as a device filter for the [BtLibDiscoverMultipleDevices](#) and [BtLibDiscoverSingleDevice](#) functions. With this filter, devices in any major device class appear in the device list. Same as `btLibCOD_Major_Mask`.

`btLibCOD_Minor_Any`

Used as a device filter for the [BtLibDiscoverMultipleDevices](#) and [BtLibDiscoverSingleDevice](#) functions.

With this filter, devices in any minor device class appear in the device list. Same as `btLibCOD_Minor_Mask`.

`btLibCOD_Minor_Comp_Any`

Used as a device filter for the [BtLibDiscoverMultipleDevices](#) and [BtLibDiscoverSingleDevice](#) functions.

When this filter is used in conjunction with `btLibCOD_Major_Computer`, all devices broadcasting themselves as computers appear in the device list. Same as `btLibCOD_Minor_Any`.

`btLibCOD_Minor_Phone_Any`

Used as a device filter for the

[BtLibDiscoverMultipleDevices](#) and [BtLibDiscoverSingleDevice](#) functions.

When this filter is used in conjunction with `btLibCOD_Major_Phone`, all devices broadcasting themselves as phones appear in the device list. Same as `btLibCOD_Minor_Any`.

`btLibCOD_Minor_LAN_Any`

Used as a device filter for the

[BtLibDiscoverMultipleDevices](#) and [BtLibDiscoverSingleDevice](#) functions.

When this filter is used in conjunction with `btLibCOD_Major_Lan_Access_Point`, all devices broadcasting themselves as LAN access points appear in the device list. Same as `btLibCOD_Minor_Any`.

## Bluetooth Library: Management

### Bluetooth Management Data Structures

---

`btLibCOD_Minor_Audio_Any`

Used as a device filter for the [`BtLibDiscoverMultipleDevices`](#) and [`BtLibDiscoverSingleDevice`](#) functions.

When this filter is used in conjunction with `btLibCOD_Major_Audio`, all devices broadcasting themselves as audio devices appear in the device list. Same as `btLibCOD_Minor_Any`.



## BtLibConnectionRoleEnum

The `BtLibConnectionRoleEnum` enum specifies all the connection roles a device can have. A device can either be a master or a slave.

```
typedef enum {
    btLibMasterRole,
    btLibSlaveRole
} BtLibConnectionRoleEnum;
```

### Value Descriptions

`btLibMasterRole` The device is a master.  
`btLibSlaveRole` The device is a slave.



## BtLibDeviceAddressType

The `BtLibDeviceAddressType` structure represents a 48-bit Bluetooth device address.

```
#define btLibDeviceAddressSize 6

typedef struct BtLibDeviceAddressType {
    UInt8 address [btLibDeviceAddressSize];
} BtLibDeviceAddressType;
```



## BtLibFriendlyNameType

The BtLibFriendlyNameType structure contains the user-friendly name of a device.

```
typedef struct BtLibFriendlyNameType {  
    UInt8 *name;  
    UInt8 nameLength;  
} BtLibFriendlyNameType,  
    *BtLibFriendlyNameTypePtr;
```

### Field Descriptions

name	Array of characters, encoded according to the UTF-8 standard, containing the user-friendly name of the device. This array is not null-terminated.
nameLength	The number of characters in the user-friendly name. The maximum size is 249 characters.



## BtLibManagementEventType

The BtLibManagementEventType structure contains detailed information regarding a management callback event. All management events have some common data. Most management events have data specific to those events. The specific data uses a union that is part of the BtLibManagementEvent data structure.

```
typedef struct _BtLibManagementEventType {  
    BtLibManagementEventEnum event;  
    Err status;  
    union {  
        ...  
    } eventData;  
} BtLibManagementEventType;
```

## Bluetooth Library: Management

### Management Callback Events

---

#### Field Descriptions

event	A BtLibManagementEventEnum enum member that indicates which management event has occurred. See <a href="#">Management Callback Events</a> .
status	Status of the event. The <a href="#">Management Callback Events</a> section gives more details about how to interpret this field for specific events.
eventData	Data associated with the event. The member of this union that is valid depends on the event. See <a href="#">Management Callback Events</a> for more information.
A BtLibManagementEventType object is passed as the first argument of the <a href="#">BtLibManagementCallback</a> callback function.	

## Management Callback Events

The management functions of the Bluetooth library support the lower levels of the Bluetooth specification, specifically the radio, baseband, and Link Manager Protocol specifications. Most of the management functions are asynchronous. In other words, they start operations and return before the operations complete. To signal the application that management operations have completed, the Bluetooth library generates *management callback events* by calling a callback function.

You specify the callback function using the [BtLibRegisterManagementNotification](#) function. When an event occurs, the callback function is called with two parameters: a pointer to a [BtLibManagementEventType](#) structure and a pointer to a user-defined structure.

The BtLibManagementEventType structure contains an event field, which indicates the reason the callback is called, a status field, which contains status information associated with the event, and a union of several structures. The member of the union that is valid depends on the event. The meaning of the events is described in the following sections.

For more information about the status field, see [Management Event Status Codes](#).

## **btLibManagementEventAccessibilityChange**

The accessibility mode of the local device has changed.

For this event, the eventData field contains the following field:

```
BtLibAccessibleModeEnum accessible;
```

This [BtLibAccessibleModeEnum](#) represents the new accessibility mode of the local device.

This event can result from calling [BtLibOpen](#), [BtLibPiconetCreate](#), or [BtLibSetGeneralPreference](#).

## **btLibManagementEventAclConnectInbound**

A remote device has established an ACL link to the local device.

For this event, the eventData field contains the following field:

```
BtLibDeviceAddressType bdAddr;
```

This [BtLibDeviceAddressType](#) contains the address of the remote device.

## **btLibManagementEventAclConnectOutbound**

An ACL link has been established with a remote device.

If the status field contains [btLibErrNoError](#), the ACL link is connected, and the remote device address can be found in the eventData field. Otherwise the connection failed and the status field indicates the reason for the failure. See [Management Event Status Codes](#) for more information.

For this event, the eventData field contains the following field:

```
BtLibDeviceAddressType bdAddr;
```

This [BtLibDeviceAddressType](#) contains the address of the remote device.

This event can result from calling [BtLibLinkConnect](#).

## **Bluetooth Library: Management**

### *Management Callback Events*

---

#### **btLibManagementEventAclDisconnect**

An ACL link has been disconnected. The status field indicates the reason the link was disconnected. See [Management Event Status Codes](#).

For this event, the eventData field contains the following field:

```
BtLibDeviceAddressType bdAddr;
```

This [BtLibDeviceAddressType](#) contain the address of the disconnected device.

This event can result from calling [BtLibLinkDisconnect](#) or [BtLibPiconetDestroy](#).

#### **btLibManagementEventAuthenticationComplete**

The authentication of a remote device has completed.

For this event, the eventData field contains the following field:

```
BtLibDeviceAddressType bdAddr;
```

This [BtLibDeviceAddressType](#) contains the address of the remote device.

If the authentication is successful, the status field contains `btLibErrNoError`. If the user cancels the passkey request, the status field contains `btLibErrCanceled`. Otherwise, the status field indicates the reason the authentication failed. See [Management Event Status Codes](#).

This event can result from calling [BtLibLinkSetState](#).

#### **btLibManagementEventEncryptionChange**

Encryption for a link has been enabled or disabled.

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

```
struct {
    BtLibDeviceAddressType bdAddr;
    Boolean enabled;
} encryptionChange;
```

### Field Descriptions

bdAddr	A <a href="#">BtLibDeviceAddressType</a> containing the address of the remote device.
enabled	true when encryption for the link has been enabled; false otherwise.

This event can result from calling [BtLibLinkSetState](#).

## btLibManagementEventInquiryCanceled

The device inquiry has been canceled because the application called [BtLibCancelInquiry](#).

## btLibManagementEventInquiryComplete

The device inquiry started with the [BtLibStartInquiry](#) function has completed.

## btLibManagementEventInquiryResult

A remote device has responded to an inquiry that was started with the [BtLibStartInquiry](#) function.

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

```
struct {
    BtLibDeviceAddressType bdAddr;
    BtLibClassOfDeviceType classOfDevice;
} inquiryResult;
```

### Field Descriptions

bdAddr	A <a href="#">BtLibDeviceAddressType</a> containing the address of the remote device.
classOfDevice	A <a href="#">BtLibClassOfDeviceType</a> representing the class of the remote device.

## btLibManagementEventLocalNameChange

The user-friendly name of the local device has changed.

## Bluetooth Library: Management

### Management Callback Events

---

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

```
struct {
    BtLibDeviceAddressType bdAddr;
    BtLibFriendlyNameType name;
} nameResult;
```

#### Field Descriptions

bdAddr	A <a href="#">BtLibDeviceAddressType</a> containing the address of the local device.
name	A <a href="#">BtLibFriendlyNameType</a> containing the new name.

This event can result from calling [BtLibOpen](#).

## btLibManagementEventModeChange

A slave has changed its mode. A slave can be in active, sniff, hold, or park mode.

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

```
struct {
    BtLibDeviceAddressType bdAddr;
    BtLibLinkModeEnum curMode;
    UInt16 interval;
} modeChange;
```

#### Field Descriptions

bdAddr	A <a href="#">BtLibDeviceAddressType</a> containing the address of the remote device.
curMode	A <a href="#">BtLibLinkModeEnum</a> indicating the new mode of remote device
interval	The time in units of 0.625 ms the remote device will stay in the new mode, if applicable. The time period is a standard time period in the Bluetooth specification.

### BtLibLinkModeEnum

The BtLibLinkModeEnum enum specifies the modes a slave can have. According to the *Specification of the Bluetooth System*, a slave can be in active, sniff, hold, or park mode. However, the Bluetooth library only supports the hold and active modes.

```
typedef enum {
    btLibSniffMode,
    btLibHoldMode,
    btLibParkMode,
    btLibActiveMode
} BtLibLinkModeEnum;
```

### Value Descriptions

`btLibActiveMode`

The slave is active.

`btLibHoldMode` The slave is in hold mode.

`btLibParkMode` The slave is in park mode. This mode is not currently supported.

`btLibSniffMode` The slave is in sniff mode. This mode is not currently supported.

### btLibManagementEventNameResult

A remote device name request has completed. If the `status` field is `btLibErrNoError`, the name is available. Otherwise, the name request failed, and the `status` field indicates the reason for the failure. See [Management Event Status Codes](#).

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

```
struct {
    BtLibDeviceAddressType bdAddr;
    BtLibFriendlyNameType name;
} nameResult;
```

### Field Descriptions

`bdAddr`

A [BtLibDeviceAddressType](#) containing the address of the remote device.

## **Bluetooth Library: Management**

### *Management Callback Events*

---

name            A [BtLibFriendlyNameType](#) containing the name of the remote device.

The [BtLibGetRemoteDeviceName](#) function is used to start a remote device name request.

## **btLibManagementEventPasskeyRequest**

A remote device has requested a passkey. Your application does not have to respond to this request—the Bluetooth library automatically handles it.

For this event, the `eventData` field contains the following field:

```
BtLibDeviceAddressType bdAddr;
```

This [BtLibDeviceAddressType](#) contains the address of the remote device.

Because a passkey can be requested during or after a link is established, consider disabling any failure timers while the passkey dialog is up. The [btLibManagementEventPasskeyRequestComplete](#) event signals the completion of the passkey entry.

## **btLibManagementEventPasskeyRequestComplete**

A passkey request has been processed. The status code for this event is set to `btLibErrNoError` if the passkey was entered or `btLibErrCanceled` if passkey entry was cancelled. Note that this event does **not** tell you that the authentication completed.

## **btLibManagementEventPiconetCreated**

The piconet has been created. This event can result from calling [BtLibPiconetCreate](#).

## **btLibManagementEventPiconetDestroyed**

The piconet has been destroyed. This event can result from calling [BtLibPiconetDestroy](#).

## **btLibManagementEventRadioState**

This event is generated when the Bluetooth radio changes state. The radio changes state when the radio is disconnected, the power is turned on or off, the radio resets, or the radio fails to initialize. The status code for this event explains why the event gets generated.

### **Status Codes**

#### `btLibErrRadioInitialized`

The Bluetooth radio has initialized successfully.  
You can now call management functions.

#### `btLibErrRadioInitFailed`

The Bluetooth radio failed to initialize. The application can assume all pending Bluetooth operations have failed. However, some pending operations will still generate events and modify memory supplied by the application.

To try to initialize the radio again, you need to close the library and reopen it.

## **Bluetooth Library: Management**

### *Management Callback Events*

---

#### `btLibErrRadioFatal`

A fatal radio error occurred. This usually signifies that the host has lost contact with the radio, for example, when the user disconnects the radio, or the device turns off. The application can assume that all pending operations have failed. However, some pending operations will still generate events and modify memory supplied by the application.

When a fatal radio error occurs, the Bluetooth stack resets the radio and tries once to reinitialize it, which generates another `btLibManagementEventRadioState` event with a status code of `btLibErrRadioInitialized`, or `btLibErrRadioInitFailed` depending on whether or not the initialization succeeded.

#### `btLibErrRadioSleepWake`

The radio was reset because the device went to sleep. The application can assume all pending operations have failed. However, some pending operations will still generate events and modify memory supplied by the application.

The Bluetooth stack resets the radio and tries once to reinitialize it, which generates another `btLibManagementEventRadioState` event with a status code of `btLibErrRadioInitialized`, or `btLibErrRadioInitFailed` depending on whether or not the initialization succeeded.

This event can result from calling [BtLibOpen](#).

## **btLibManagementEventRoleChange**

The master and slave devices for a link have switched roles.

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

```
struct {
    BtLibDeviceAddressType bdAddr;
    BtLibConnectionRoleEnum newRole;
} roleChange;
```

### Field Descriptions

<code>bdAddr</code>	A <a href="#">BtLibDeviceAddressType</a> containing the address of the remote device.
<code>newRole</code>	A <a href="#">BtLibConnectionRoleEnum</a> representing the new role of the local device.

## Management Event Status Codes

When a management event is generated, the `status` field of the associated [BtLibManagementEventType](#) provides information about why the event occurred. The following status codes can occur with a management event.

<code>btLibErrNoError</code>	Success.
<code>btLibMeStatusAuthenticateFailure</code>	Authentication failure
<code>btLibMeStatusCommandDisallowed</code>	Command disallowed
<code>btLibMeStatusConnnectionTimeout</code>	Connection timeout
<code>btLibMeStatusHardwareFailure</code>	Hardware Failure
<code>btLibMeStatusHostTimeout</code>	Host timeout
<code>btLibMeStatusInvalidHciParam</code>	Invalid HCI command parameters
<code>btLibMeStatusInvalidLmpParam</code>	Invalid LMP Parameters

## **Bluetooth Library: Management**

### *Management Event Status Codes*

---

btLibMeStatusLimitedResources  
Host rejected due to limited resources

btLibMeStatusLmpResponseTimeout

btLibMeStatusLmpTransdCollision

btLibMeStatusLmpPduNotAllowed

btLibMeStatusLocalTerminated  
Terminated by local host

btLibMeStatusLowResources  
Other end terminated due to low resources

btLibMeStatusMaxAclConnections  
Max number of ACL connections to a device

btLibMeStatusMaxConnections  
Max number of connections

btLibMeStatusMaxScoConnections  
Max number of SCO connections to a device

btLibMeStatusMemoryFull  
Memory full

btLibMeStatusMissingKey  
Missing key

btLibMeStatusNoConnection  
No connection

btLibMeStatusPageTimeout  
Page timeout

btLibMeStatusPairingNotAllowed  
Pairing not allowed

btLibMeStatusPersonalDevice  
Host rejected (remote is personal device)

btLibMeStatusPowerOff  
Other end terminated (about to power off)

btLibMeStatusRepeatedAttempts  
Repeated attempts

btLibMeStatusRoleChangeNotAllowed  
Change not allowed

btLibMeStatusScoAirModeRejected  
SCO Air Mode Rejected

btLibMeStatusScoIntervalRejected  
SCO Interval Rejected

btLibMeStatusScoOffsetRejected  
SCO Offset Rejected

btLibMeStatusSecurityError  
Host rejected due to security reasons

btLibMeStatusUnknownHciCommand  
Unknown HCI Command

btLibMeStatusUnknownLmpPDU  
Unknown LMP PDU

btLibMeStatusUnspecifiedError  
Unspecified Error

btLibMeStatusUnsupportedFeature  
Unsupported feature or parameter value

btLibMeStatusUnsupportedLmpParam  
Unsupported LMP Parameter Value

btLibMeStatusUnsupportedRemote  
Unsupported Remote Feature

btLibMeStatusUserTerminated  
Other end terminated (user)

## Library Management Functions

This section describes the general Bluetooth library management functions.

## Bluetooth Library: Management

### *Library Management Functions*

---



**New**

## BtLibClose

**Purpose** Close the Bluetooth library.

**Declared In** BtLib.h

**Prototype** Err BtLibClose (UInt16 btLibRefNum)

**Parameters** -> btLibRefNum Reference number for the Bluetooth library.

**Result** Returns btLibErrNoError if the library successfully closes.  
Returns btLibErrNotOpen if the referenced Bluetooth library was not open.

**Comments** Applications must call this function when they no longer need the Bluetooth library. If the Bluetooth library open count is one, this function closes existing connections, saves the current accessibility mode, sets the accessible mode according to the preferences panel, and shuts down the library. If the Bluetooth library open count is greater than one, this function decrements the open count.

**See Also** [BtLibOpen](#)



**New**

## BtLibOpen

**Purpose** Open and initialize the Bluetooth library.

**Declared In** BtLib.h

**Prototype** Err BtLibOpen (UInt16 btLibRefNum,  
Boolean allowStackToFail)

**Parameters** -> btLibRefNum Reference number for the Bluetooth library.

-> `allowStackToFail`

If true, opens the library even if the stack or radio fails to initialize. Otherwise, does not open the library if the stack or radio fails to initialize.

**Result** Returns one of the following values:

`btLibErrNoError`  
Success.

`btLibErrAlreadyOpen`  
This status code is not really an error. It is returned if the library was already open. In this case, the open count is incremented.

`btLibErrInUseByService`  
The library is currently in use by a Bluetooth service.

`btLibErrOutOfMemory`  
Not enough memory available to open the library.

`btLibErrRadioInitFailed`  
The Bluetooth stack or radio could not be initialized. If `allowStackToFail` is true, the library still opens after this error occurs.

Applications must call this function before using the Bluetooth library. If the Bluetooth library is not already open, `BtLibOpen` opens the library, initializes it, and starts up the protocol stack component of the library. Otherwise it increments its open count.

The `allowStackToFail` parameter allows the library to be opened even if the Bluetooth stack or radio fails to initialize. It is useful for applications that only want to use the Bluetooth library's utility functions but not the radio. However, any application that needs to communicate with the radio must set `allowStackToFail` to false.

This function generates three events: a `btLibManagementEventRadioState` event with a status of `btLibErrRadioInitialized`, a `btLibManagementEventLocalNameChange` event indicating the

## Bluetooth Library: Management

### Management Functions

---

local name of the device, and a [`btLibManagementEventAccessibilityChange`](#) event indicating the accessibility of the device.

**See Also** [BtLibClose](#)

## Management Functions

The management functions handle the lower levels of the Bluetooth specification, specifically the radio, baseband, and Link Manager Protocol specifications. These functions perform tasks that include discovering devices, working with Asynchronous Connectionless (ACL) links and piconets, and maintaining global settings for the Bluetooth library.



### **BtLibCancelInquiry**

---

**Purpose** Cancel a Bluetooth inquiry in process.

**Declared In** `BtLib.h`

**Prototype** `Err BtLibCancelInquiry (UInt16 btLibRefNum)`

**Parameters** `-> btLibRefNum` Reference number for the Bluetooth library.

**Result** Returns one of the following values:

`btLibErrNoError`

The inquiry process was canceled before it started.

`btLibErrPending`

The cancellation is pending. When it succeeds, notification will be provided through a management callback event.

`btLibErrInProgress`

The inquiry is already being canceled.

`btLibErrNotInProgress`

No inquiry is in progress to be canceled.

`btLibErrNotOpen`

The referenced Bluetooth library is not open.

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

**Comments**

The function cancels inquiries initiated by [BtLibStartInquiry](#). The [btLibManagementEventInquiryCanceled](#) callback event indicates that the cancellation has completed.

A Bluetooth discovery initiated using either [BtLibDiscoverSingleDevice](#) or [BtLibDiscoverMultipleDevices](#) cannot be canceled with this function. Only the user can cancel these inquiries by tapping the Cancel button.

**See Also** [BtLibStartInquiry](#)



---

## BtLibDiscoverMultipleDevices

**Purpose** Discover all available devices, present them in the user interface, and allow the user to select one or more of these devices.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibDiscoverMultipleDevices
(UInt16 btLibRefNum, Char *instructionTxt,
Char *buttonTxt,
BtLibClassOfDeviceType *deviceFilterList,
UInt8 deviceFilterListLen,
UInt8 *numDevicesSelected, Boolean addressAsName,
Boolean showLastList)
```

**Parameters** -> `btLibRefNum` Reference number for the Bluetooth library.

## **Bluetooth Library: Management**

### *Management Functions*

---

-> instructionTxt  
Text displayed at the top of the selection box.  
Pass NULL to display the default text. The  
default text is "Select one or more devices."

-> buttonTxt Text for the OK button. Pass NULL to display  
the default text. The default button text is "OK"

-> deviceFilterList Array of [BtLibClassOfDeviceTypes](#). This  
function presents to the user only the remote  
devices whose class matches a class in this list.  
If deviceFilterList is NULL, this function  
presents to the user all discovered devices.

->deviceFilterListLen  
Number of elements in deviceFilterList.

<- numDevicesSelected  
Number of selected devices. To obtain the  
actual device list, use the  
[BtLibGetSelectedDevices](#) function.

-> addressAsName  
If true, display the Bluetooth addresses of the  
remote devices instead of their names. This  
option is available for debugging purposes.

-> showLastList  
If true, causes all other parameters to be  
ignored and displays the same list as the  
previous call to  
[BtLibDiscoverMultipleDevices](#).

**Result** Returns one of the following values:

[btLibErrNoError](#)

Success

[btLibErrCanceled](#)

User canceled discovery.

[btLibErrNotOpen](#)

The referenced Bluetooth library is not open.

**btLibErrStackNotOpen**

The Bluetooth stack failed to initialize when the library was opened.

**Comments**

This blocking call performs a full discovery for an application, including name and feature retrieval and testing. This function takes over the UI and presents a choice box to the user, allowing the user to select multiple devices from the list of devices that were discovered. This function does not return until the user chooses one or more devices, or cancels.

Setting the `showLastList` parameter to `true` allows you to present to the user the list of devices displayed in the previous call to `BtLibDiscoverMultipleDevices` or `BtLibDiscoverSingleDevice`. This feature can be useful because a full discovery process takes approximately ten seconds. The cached device list remains valid even after you close the library, allowing other Bluetooth applications to use it.

Note that `BtLibStartInquiry` overwrites the cached device list. If you are using the `showLastList` feature, you should avoid calling `BtLibStartInquiry` between calls to `BtLibDiscoverMultipleDevices` or `BtLibDiscoverSingleDevice`.

Use `BtLibGetSelectedDevices` to retrieve the list of devices that the user selected.

**See Also**

`BtLibGetSelectedDevices`, `BtLibDiscoverSingleDevice`

## Bluetooth Library: Management

### Management Functions

---



New

## BtLibDiscoverSingleDevice

**Purpose** Discover all available devices, present them in the user interface, and allow the user to select one of these devices.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibDiscoverSingleDevice
    (UInt16 btLibRefNum, Char *instructionTxt,
     BtLibClassOfDeviceType *deviceFilterList,
     UInt8 deviceFilterListLen,
     BtLibDeviceAddressType *selectedDeviceP,
     Boolean addressAsName, Boolean showLastList)
```

**Parameters** -> btLibRefNum Reference number for the Bluetooth library.

-> instructionTxt  
Text displayed at the top of the selection box.  
Pass NULL to display the default text. The  
default text is "Select a device:"

-> deviceFilterList  
Array of [BtLibClassOfDeviceTypes](#). This  
function displays only the remote devices  
whose class matches a class in this list. If  
deviceFilterList is NULL, this function  
displays all discovered devices.

-> deviceFilterListLen  
Number of elements in deviceFilterList.

<- selectedDeviceP  
Pointer to a [BtLibDeviceAddressType](#)  
where this function stores the address of the  
device the user selects. You need to allocate this  
space before calling this function.

-> addressAsName  
If true, display the Bluetooth addresses of the  
remote devices instead of their names. This  
option is available for debugging purposes.

-> showLastList

If true, causes all other parameters to be ignored and displays the same list as the previous call to BtLibDiscoverSingleDevice.

**Result** Returns one of the following values:

btLibErrNoError  
Success

btLibErrCanceled  
User canceled the discovery.

btLibErrNotOpen  
The referenced Bluetooth library is not open.

btLibErrStackNotOpen  
The Bluetooth stack failed to initialize when the library was opened.

**Comments**

This blocking call performs a full discovery for an application, including name and feature retrieval and testing. This function takes over the UI and presents a choice box to the user, allowing the user to select a device from the list of devices that were discovered. This function does not return until the user chooses a device or cancels.

Setting the showLastList parameter to true allows you to present to the user the list of devices displayed in the previous call to BtLibDiscoverSingleDevice or

[BtLibDiscoverMultipleDevices](#). This feature can be useful because a full discovery process takes approximately ten seconds. The cached device list remains valid even after you close the library, allowing other Bluetooth applications to use it.

Note that [BtLibStartInquiry](#) overwrites the cached device list. If you are using the showLastList feature, you should avoid calling BtLibStartInquiry between calls to BtLibDiscoverSingleDevice or BtLibDiscoverMultipleDevices.

**See Also** [BtLibDiscoverMultipleDevices](#)



New

## BtLibGetGeneralPreference

**Purpose** Get one of the general management preferences.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibGetGeneralPreference  
(UInt16 btLibRefNum, BtLibGeneralPrefEnum pref,  
void *prefValue, UInt16 prefValueSize)
```

**Parameters**

-> btLibRefNum	Reference number for the Bluetooth library.
-> pref	General preference to get.
<- prefValue	Pointer to a buffer to hold the value of the preference. You must allocate this buffer. This parameter must not be NULL.
-> prefValueSize	Size, in bytes, of the prefValue buffer. You must set this size so it matches the size of the retrieved preference.

**Result** Returns one of the following values:

btLibErrNoError

Success.

btLibErrNotOpen

The referenced Bluetooth library is not open.

btLibErrParamError

One or more parameters is invalid. Be sure that the prefValueSize parameter matches the size of the preference value.

btLibErrStackNotOpen

The Bluetooth stack failed to initialize when the library was opened.

**Comments** Specify the preference with a member of the BtLibGeneralPreferenceEnum.

### **BtLibGeneralPreferenceEnum**

The BtLibGeneralPreferenceEnum enum specifies the general preferences that can be accessed using the [BtLibSetGeneralPreference](#) and [BtLibGetGeneralPreference](#) functions.

```
typedef enum {
    btLibPref_Name,
    btLibPref_UnconnectedAccessible,
    btLibPref_CurrentAccessible,
    btLibPref_LocalClassOfDevice,
    btLibPref_LocalDeviceAddress
} BtLibGeneralPrefEnum;
```

#### **Value Descriptions**

**btLibPref\_CurrentAccessible**

This preference is a [BtLibAccessibleModeEnum](#) indicating the current accessibility mode of the local device.

**btLibPref\_LocalClassOfDevice**

This preference is a [BtLibClassOfDeviceType](#) indicating the class of the local device. You should never set this preference.

**btLibPref\_Name**

This preference is a [BtLibFriendlyNameType](#) containing the user-friendly name of the local device. If you retrieve this preference, you also need to allocate a buffer and set the BtLibFriendlyNameType's name and nameLength fields to the buffer pointer and buffer length, respectively.

You should never set this preference.

## Bluetooth Library: Management

### Management Functions

---

`btLibPref_UnconnectedAccessible`

This preference is a

[BtLibAccessibleModeEnum](#) indicating the accessibility mode of the local device when it is unconnected. You should never set this preference.

**See Also** [BtLibSetGeneralPreference](#)



## BtLibGetRemoteDeviceName

**Purpose** Get the name of the remote device with the specified address.

**Declared In** BtLib.h

**Prototype** Err BtLibGetRemoteDeviceName (UInt16 btLibRefNum,  
BtLibDeviceAddressTypePtr remoteDeviceP,  
BtLibFriendlyNameType \*nameP,  
BtLibGetNameEnum retrievalMethod)

**Parameters** -> `btLibRefNum` Reference number for the Bluetooth library.

-> `remoteDeviceP` Pointer to a [BtLibDeviceAddressType](#) containing the address of the device whose name is desired.

<-> `nameP` Pointer to a [BtLibFriendlyNameType](#) structure in which to store the results of the lookup. You must allocate this structure and the name buffer it points to. You also must specify the size of the buffer in the `nameLength` field of the structure. When the function returns, the `nameLength` field contains the actual length of the name. This parameter must not be NULL.

-> retrievalMethod

Method used to retrieve the user-friendly remote device name. See [BtLibGetNameEnum](#).

**Result** Returns one of the following values:

btLibErrNoError

The name structure was successfully retrieved from the cache. No notification event will be generated.

btLibErrBusy

There is already a name request pending.

btLibErrNotOpen

The referenced Bluetooth library is not open.

btLibErrPending

The results will be returned through a notification.

btLibErrStackNotOpen

The Bluetooth stack failed to initialize when the library was opened.

**Comments**

The Bluetooth library maintains a cache of 50 device names. If the retrievalMethod parameter is btLibCachedThenRemote, this function first checks the cache for a name. If the name is in the cache, the value is returned immediately in the nameP parameter. If the name is not in the cache, the function queries the remote device for its name, forming a temporary ACL connection if one is not already in place. In this case, the function returns btLibErrPending and generates a [BtLibManagementEventNameResult](#) event when the name is available.

Other values of the retrievalMethod parameter can instruct this function to look for the name only in the cache or only on the remote device. See [BtLibGetNameEnum](#) for more information.

**BtLibGetNameEnum**

The BtLibGetNameEnum enum specifies whether to retrieve a device name from the cache, the remote device, or both.

## Bluetooth Library: Management

### Management Functions

---

```
typedef enum {
    btLibCachedThenRemote,
    btLibCachedOnly,
    btLibRemoteOnly
} BtLibGetNameEnum;
```

#### **Value Descriptions**

`btLibCachedOnly`

Look for a name in the cache. If the name is not in the cache, fail.

`btLibCachedThenRemote`

Look for a name in the cache. If the name is not in the cache, ask the remote device.

`btLibRemoteOnly`

Ignore any cached names and ask the remote device for its name.



## BtLibGetSelectedDevices

---

**Purpose** Get the list of devices selected during the last call to [BtLibDiscoverMultipleDevices](#).

**Declared In** BtLib.h

**Prototype** Err BtLibGetSelectedDevices (UInt16 btLibRefNum,  
BtLibDeviceAddressType \*selectedDeviceArray,  
UInt8 arraySize, UInt8 \*numDevicesReturned)

**Parameters**

-> btLibRefNum	Reference number for the Bluetooth library.
<->selectedDeviceArray	Array into which the results of the <a href="#"><u>BtLibDiscoverMultipleDevices</u></a> function should be placed. You must allocate this array of <a href="#"><u>BtLibDeviceAddressTypes</u></a> .
-> arraySize	Number of elements in the selectedDeviceArray you allocated.

<- numDevicesReturned  
Number of results placed in  
selectedDeviceArray.

**Result** Returns btLibErrNoError if the query is successful. Returns btLibErrNotOpen if the referenced Bluetooth library is not open or btLibErrStackNotOpen if the Bluetooth stack failed to initialize when the library was opened.

**Comments** No callback events.

**See Also** [BtLibDiscoverMultipleDevices](#)



## New **BtLibLinkConnect**

---

**Purpose** Create a Bluetooth Asynchronous Connectionless (ACL) link.

**Declared In** BtLib.h

**Prototype** Err BtLibLinkConnect (UInt16 btLibRefNum,  
BtLibDeviceAddressTypePtr remoteDeviceP)

**Parameters** -> btLibRefNum Reference number for the Bluetooth library.  
-> remoteDeviceP Pointer to the a [BtLibDeviceAddressType](#) containing the address of the remote device.

**Result** Returns one of the following values:

btLibErrPending  
The results will be returned through a callback event.  
btLibErrAlreadyConnected  
The device is already in a pre-existing connection and cannot create a new connection.

## Bluetooth Library: Management

### Management Functions

---

`btLibErrBluetoothOff`

The Bluetooth radio is off. The user can turn the radio on and off with a setting in the preferences panel.

`btLibErrBusy`

A piconet is currently being created or destroyed.

`btLibErrNotOpen`

The referenced Bluetooth library is not open.

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

`btLibErrTooMany`

Cannot create another ACL link because the maximum allowed number has already been reached.

#### Comments

An ACL link is a packet-switched physical level connection between two devices that is needed before the devices can form a RFCOMM or L2CAP connection.

When the connection is established or if it fails to be established, the [btLibManagementEventAclConnectOutbound](#) event is generated.

#### See Also

[BtLibLinkDisconnect](#)



---

## BtLibLinkDisconnect

**Purpose** Disconnect an existing ACL Link.

**Declared In** BtLib.h

**Prototype** Err BtLibLinkDisconnect (UInt16 btLibRefNum,  
BtLibDeviceAddressTypePtr remoteDeviceP)

**Parameters** -> btLibRefNum Reference number for the Bluetooth library.

-> remoteDeviceP

Pointer to a [BtLibDeviceAddressType](#) containing the address of the remote device.

**Result** Returns one of the following values:

btLibErrNoError

The connection attempt was canceled before it started. No event is generated.

btLibErrPending

When the link actually disconnects, a [btLibManagementEventAclDisconnect](#) callback event is generated.

btLibErrBusy

Can't disconnect the link because the piconet is being destroyed.

btLibErrNoConnection

No link to the specified device exists.

btLibErrNotOpen

The referenced Bluetooth library is not open.

btLibErrStackNotOpen

The Bluetooth stack failed to initialize when the library was opened.

**Comments**

When the link disconnects, a

[btLibManagementEventAclDisconnect](#) event is generated.

**See Also**

[BtLibLinkConnect](#)

## Bluetooth Library: Management

### Management Functions

---



#### New **BtLibLinkGetState**

**Purpose** Get the state of an ACL link.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibLinkGetState(UInt16 btLibRefNum,
                      BtLibDeviceAddressTypePtr remoteDeviceP,
                      BtLibLinkPrefsEnum pref, void *linkState,
                      UInt16 linkStateSize)
```

**Parameters**

-> btLibRefNum Reference number for the Bluetooth library.

-> remoteDeviceP  
Pointer to a [BtLibDeviceAddressType](#) containing the address of the remote device. This address identifies the ACL link.

-> pref Link preference to retrieve. See [BtLibLinkPrefsEnum](#).

<- linkState Pointer to a buffer to store the value of the preference. You must allocate this buffer. This parameter must not be NULL. See [BtLibLinkPrefsEnum](#) for more information.

-> linkStateSize  
Size, in bytes, of linkState buffer. This size must match the size of the retrieved preference.

**Result** Returns one of the following values:

**btLibErrNoError**

Success. The linkState variable has been filled in.

**btLibErrNoAclLink**

No link to the specified remote device exists.

**btLibErrNotOpen**

The referenced Bluetooth library is not open.

`btLibErrParamError`

The `linkStateSize` parameter is not same as the size of the preference value.

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

**Comments** See the [BtLibLinkSetState](#) function description for a list of the link states preferences.

**See Also** [BtLibLinkSetState](#)



---

## BtLibLinkSetState

**Purpose** Set the state of an ACL link

**Declared In** BtLib.h

**Prototype** Err BtLibLinkSetState (UInt16 btLibRefNum,  
BtLibDeviceAddressTypePtr remoteDeviceP,  
BtLibLinkPrefsEnum pref, void \*linkState,  
UInt16 linkStateSize)

**Parameters** -> `btLibRefNum` Reference number for the Bluetooth library.

-> `remoteDeviceP` The address of the remote device. This address identifies the ACL link.

-> `pref` Link preference to set. See [BtLibLinkPrefsEnum](#).

-> `linkState` Pointer to the value of the preference. This parameter must not be NULL. See [BtLibLinkPrefsEnum](#).

## Bluetooth Library: Management

### Management Functions

---

-> linkStateSize

Size, in bytes, of the linkState value.

**Result** Returns one of the following values:

btLibErrPending

The results will be returned through a callback event.

btLibErrFailed An attempt was made to encrypt a link before authenticating it.

btLibErrNoAclLink

No link to the specified remote device exists.

btLibErrNotOpen

The referenced Bluetooth library is not open.

btLibErrParamError

The preference cannot be set or linkStateSize is invalid.

btLibErrStackNotOpen

The Bluetooth stack failed to initialize when the library was opened.

### Comments

Applications use this function to set the state of an ACL link. This function may generate events depending on the preference you change. The [btLibManagementEventAuthenticationComplete](#) event indicates the link authentication has completed. The [btLibManagementEventEncryptionChange](#) indicates that the encryption has been enabled or disabled.

### BtLibLinkPrefsEnum

The BtLibLinkPrefsEnum enum specifies the link state preferences that can be accessed with the [BtLibLinkSetState](#) and [BtLibLinkGetState](#) functions.

```
typedef enum {
    btLibLinkPref_Authenticated,
    btLibLinkPref_Encrypted,
    btLibLinkPref_LinkRole
} BtLibLinkPrefsEnum;
```

### **Value Descriptions**

`btLibLinkPref_Authenticated`

This preference is a Boolean and indicates whether the link has been authenticated or not.

`btLibLinkPref_Encrypted`

This preference is a Boolean and indicates whether the link is encrypted or not.

`btLibLinkPref_LinkRole`

This preference is a [BtLibConnectionRoleEnum](#) and indicates whether the remote device is a master or a slave. You cannot set this preference but you can get its value.

**See Also** [BtLibLinkGetState](#)



### **BtLibPiconetCreate**

---

**Purpose** Create a piconet or reconfigure an existing piconet so the local device is the master.

**Declared In** BtLib.h

**Prototype** Err BtLibPiconetCreate (UInt16 btLibRefNum,  
Boolean unlockInbound, Boolean discoverable)

**Parameters** -> `btLibRefNum` Reference number for the Bluetooth library.

-> `unlockInbound`

If true, the piconet accepts inbound connections. Otherwise, the piconet only allows outbound connections.

## Bluetooth Library: Management

### Management Functions

---

-> discoverable

If `true`, configures the radio to be discoverable. In other words, the radio responds to inquiries. If `false`, configures the radio to be only connectable. In other words, only devices that know the radio's Bluetooth device address can connect to it. This parameter is ignored if `unlockInbound` is `false`.

**Result** Returns one of the following values:

`btLibErrNoError`

Successfully created the piconet with the local device as the master. No callback event is generated.

`btLibErrPending`

An existing ACL link needs to switch roles before this operation can complete.

`btLibErrFailed` A piconet already exists.

`btLibErrInProgress`

Another piconet is currently being created.

`btLibErrNotOpen`

The referenced Bluetooth library is not open.

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

#### Comments

If the accessibility of the radio changes due to this operation, a `btLibManagementEventAccessibilityChange` event is generated. When the piconet is created, or if the piconet fails to be created, a `btLibManagementEventPiconetCreated` event is generated. The `status` field of the `BtLibManagementEventType` structure accompanying the event indicates whether the piconet was created or not.

#### See Also

[BtLibPiconetDestroy](#), [BtLibPiconetUnlockInbound](#),  
[BtLibPiconetLockInbound](#)



## New **BtLibPiconetDestroy**

**Purpose** Destroy the piconet by disconnecting links to all devices and removing all restrictions on whether the local device is a master or a slave.

**Declared In** BtLib.h

**Prototype** Err BtLibPiconetDestroy (UInt16 btLibRefNum)

**Parameters** -> btLibRefNum Reference number for the Bluetooth library.

**Result** Returns one of the following values:

btLibErrNoError

Successfully destroyed the piconet. A [btLibManagementEventPiconetDestroy ed](#) event is not generated.

btLibErrPending

The piconet is being destroyed, and a [btLibManagementEventPiconetDestroy ed](#) event will be generated when the operation succeeds or fails.

btLibErrBusy

The piconet is already in the process of being destroyed.

btLibErrNoPiconet

No piconet exists to be destroyed.

btLibErrNotOpen

The referenced Bluetooth library is not open.

btLibErrStackNotOpen

The Bluetooth stack failed to initialize when the library was opened.

**Comments** A [btLibManagementEventAclDisconnect](#) event is generated for each ACL link that is disconnected. When the piconet is successfully destroyed or fails to be destroyed, a

## Bluetooth Library: Management

### Management Functions

---

`btLibManagementEventPiconetDestroyed` is generated. The status field of the `BtLibManagementEventType` structure accompanying the event indicates whether the piconet was destroyed or not.

**See Also** [BtLibPiconetCreate](#)



**New**

## BtLibPiconetLockInbound

**Purpose** Prevent remote devices from creating ACL links into the piconet.

**Declared In** BtLib.h

**Prototype** Err BtLibPiconetLockInbound (UInt16 btLibRefNum)

**Parameters** -> btLibRefNum Reference number for the Bluetooth library.

**Result** Returns one of the following values:

`btLibErrNoError`

Success.

`btLibErrBusy`

The piconet is in the process of being destroyed.

`btLibErrNoPiconet`

No piconet exists.

`btLibErrNotOpen`

The referenced Bluetooth library is not open.

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

**Comments** After locking inbound connections, outbound connections are still allowed. Locking inbound connections maximizes the bandwidth for members of the piconet to transmit data to each other.

**See Also** [BtLibPiconetUnlockInbound](#)



## New **BtLibPiconetUnlockInbound**

**Purpose** Allow remote devices to create ACL links into the piconet.

**Declared In** BtLib.h

**Prototype** Err BtLibPiconetUnlockInbound  
(UInt16 btLibRefNum, Boolean discoverable)

**Parameters** -> btLibRefNum Reference number for the Bluetooth library.

-> discoverable  
If true, configures the radio to be discoverable.  
In other words, the radio responds to inquiries.  
If false, configures the radio to be only  
connectable. In other words, only devices that  
know the radio's Bluetooth device address can  
connect to it.

**Result** Returns one of the following values:

btLibErrNoError

Success.

btLibErrBusy

The piconet is in the process of being  
destroyed.

btLibErrNoPiconet

No piconet exists.

btLibErrNotOpen

The referenced Bluetooth library is not open.

btLibErrStackNotOpen

The Bluetooth stack failed to initialize when the  
library was opened.

## Bluetooth Library: Management

### Management Functions

---

**Comments** Allowing inbound connections lowers the bandwidth available to transmit data between members of the piconet because the radio must periodically scan for incoming links.

**See Also** [BtLibPiconetLockInbound](#)



## BtLibRegisterManagementNotification

**Purpose** Register a callback function to process events generated by management functions.

**Declared In** BtLib.h

**Prototype** Err BtLibRegisterManagementNotification  
(UInt16 btLibRefNum,  
BtLibManagementProcPtr callbackP, UInt32 refCon)

**Parameters**

-> btLibRefNum	Reference number for the Bluetooth library.
-> callbackP	Pointer to a callback procedure to register. This pointer must not be NULL. See <a href="#">BtLibManagementCallback</a> for more information.
-> refCon	Application-defined data to pass to the event handler.

**Result** Returns one of the following values:

btLibErrNoError	Success.
btLibErrAlreadyRegistered	The callback has already been registered with the management entity.
btLibErrNotOpen	The referenced Bluetooth library is not open.
btLibErrParamError	One or more parameters is invalid.

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

`btLibErrTooMany`

There is no space available to store the callback. The management entity allows a maximum of 16 callbacks to be registered at a time.

**Comments**

The management functions are asynchronous. That is, they return immediately and generate events when the task actually completes at a later time. To handle these events, you need to define a callback function with the same prototype as [BtLibManagementCallback](#). Then you need to register your callback function using `BtLibRegisterManagementNotification`. For examples of the callback events your callback function needs to handle, see the [Management Callback Events](#) section.

Applications should unregister their management callbacks before closing the Bluetooth library to prevent the callback table from overflowing. The callback table holds a maximum of 16 entries.

**See Also**

[BtLibUnregisterManagementNotification](#)



---

## BtLibSetGeneralPreference

**Purpose** Set one of the general management preferences.

**Declared In** `BtLib.h`

**Prototype** `Err BtLibSetGeneralPreference  
(UInt16 btLibRefNum, BtLibGeneralPrefEnum pref,  
void *prefValue, UInt16 prefValueSize)`

**Parameters**

-> <code>btLibRefNum</code>	Reference number for the Bluetooth library.
-> <code>pref</code>	General preference to set. See <a href="#">BtLibGeneralPreferenceEnum</a> .

## Bluetooth Library: Management

### Management Functions

---

-> `prefValue` Pointer to the value of the preference. This parameter must not be NULL. See [BtLibGeneralPreferenceEnum](#).

-> `prefValueSize`  
Size, in bytes, of `prefValue`.

**Result** Returns one of the following values:

`btLibErrNoError`  
Success.

`btLibErrPending`  
The results will be returned through a notification.

`btLibErrNotOpen`  
The referenced Bluetooth library is not open.

`btLibErrParamError`  
One or more parameters is invalid. Be sure that `prefValueSize` matches the size of the preference value.

`btLibErrStackNotOpen`  
The Bluetooth stack failed to initialize when the library was opened.

**Comments** See the [BtLibGetGeneralPreference](#) function description for a list of the preferences.

This function may generate events depending on the preference you change. The [btLibManagementEventAccessibilityChange](#) event indicates that the accessibility of the local device has changed.

**See Also** [BtLibGetGeneralPreference](#)



## New **BtLibStartInquiry**

**Purpose** Start a Bluetooth inquiry.

**Declared In** BtLib.h

**Prototype** Err BtLibStartInquiry (UInt16 btLibRefNum,  
                  UInt8 timeout, UInt8 maxResp)

**Parameters** -> btLibRefNum Reference number for the Bluetooth library.

-> timeout Time, in seconds, this inquiry is allowed to take. If the inquiry does not complete within this time, it is canceled. The actual time is rounded to the nearest multiple of 1.28 seconds. If you specify a timeout period larger than 60 seconds, this function acts as if you specified a timeout period of 60 seconds. If this parameter is 0, the timeout period defaults to 10.24 seconds as specified in the Generic Access Profile.

-> maxResp Maximum number of responses the inquiry accepts. Responses are not guaranteed to be unique.

**Result** Returns one of the following values:

btLibErrPending

The results will be returned through callback events.

btLibErrBluetoothOff

The Bluetooth radio is off. The user can turn the radio on and off with a setting in the preferences panel.

btLibErrInProgress

Another inquiry is already in progress.

## Bluetooth Library: Management

### Management Functions

---

`btLibErrNotOpen`

The referenced Bluetooth library is not open.

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

#### Comments

The function performs a low-level Bluetooth inquiry, as opposed to a full device discovery. Specifically, inquiries started with this function only return the Bluetooth address and the class of the discovered device. This function does not have a user interface.

Every time a device is discovered, a `btLibManagementEventInquiryResult` callback event is generated. When the inquiry is complete, a `btLibManagementEventInquiryComplete` callback event is generated. If the application calls `BtLibCancelInquiry`, a `btLibManagementEventInquiryCanceled` callback event is generated.

#### See Also

[BtLibCancelInquiry](#)



## BtLibUnregisterManagementNotification

**Purpose** Unregister a previously registered management callback.

**Declared In** `BtLib.h`

**Prototype** `Err BtLibUnregisterManagementNotification  
(UInt16 btLibRefNum,  
BtLibManagementProcPtr callbackP)`

**Parameters** `-> btLibRefNum` Reference number for the Bluetooth library.  
`-> callbackP` Pointer to the callback procedure to unregister.  
This pointer must not be NULL.

**Result** Returns one of the following values:

btLibErr.NoError	Success.
btLibErr.Error	The callback referenced by <code>callbackP</code> has not been registered.
btLibErr.NotOpen	The referenced Bluetooth library is not open.
btLibErr.ParamError	One or more parameters are invalid.
btLibErr.StackNotOpen	The Bluetooth stack failed to initialize when the library was opened.

**Comments** Applications should unregister their management callbacks before closing the library to prevent the callback table from overflowing. The callback table holds a maximum of 16 entries.

**See Also** [BtLibRegisterManagementNotification](#)

## Application-Defined Functions

This section describes the callback functions that handle management events. These functions are supplied by the developer and can be named anything.



New

## BtLibManagementCallback

**Purpose** Signal the result of a Bluetooth management event. When the event takes place, this callback function is called.

**Declared In** BtLibTypes.h

**Prototype**

```
void (*BtLibManagementProcPtr)
(BtLibManagementEventType *mEvent, UInt32 refCon)
```

**Parameters**

-> mEvent	<a href="#">BtLibManagementEventType</a> structure containing the event parameters.
-> refCon	General purpose integer which you can use to hold application-specific information. When you register the callback with the <a href="#">BtLibRegisterManagementNotification</a> function, you can specify a value to pass to this parameter.

**Result** Returns nothing.

**Comments** The event and status of the event are in the mEvent structure. See [Management Callback Events](#) for more information.

You must register this function using the [BtLibRegisterManagementNotification](#) function before it starts receiving events.

# Bluetooth Library: Sockets and Service Discovery

---

The Bluetooth library uses sockets to represent L2CAP, RFCOMM, and SDP connections. This chapter presents reference material for the socket and SDP support provided by the Bluetooth library API:

## Socket-Related Data Structures

This section lists some of the more important types used by the Bluetooth library.

## Socket Callback Events

The Bluetooth library socket API supports L2CAP, RFCOMM, and SDP, the upper protocols of the Bluetooth specification. As with the management functions, the socket functions are mostly asynchronous. To signal to the application that socket operations have completed, the Bluetooth library generates *socket callback events* by calling a callback function.

## Socket Disconnection Error Codes

In addition to the standard error codes that can accompany socket events, the status codes accompanying the disconnection and connection events can have the additional values enumerated in this section.

## Socket Functions

The functions in this section perform general socket tasks and tasks related to L2CAP and RFCOMM sockets. The functions specific to SDP sockets are in the next section.

## **Bluetooth Library: Sockets and Service Discovery**

### *Socket-Related Data Structures*

---

#### [Service Discovery Protocol Functions](#)

This section describes functions and macros specific to the Bluetooth Service Discovery Protocol (SDP).

#### [Application-Defined Functions](#)

This section describes the callback functions that handle socket events.

The header file `BtLib.h` declares the Bluetooth library functions and macros. The header file `BtLibTypes.h` declares the data structures that you use with those functions and macros. For more information about using the Bluetooth library, see the *Palm OS Programmer's Companion Supplement: Bluetooth*.

## **Socket-Related Data Structures**

This section lists some of the more important types used by the Bluetooth library.



### **BtLibL2CapPsmType**

The `BtLibL2CapPsmType` type represents a Protocol and Server Multiplexer (PSM) value. See the “Logical Link and Adaptation Protocol Specification” chapter of the *Specification of the Bluetooth System* for more information about PSM values. The Bluetooth library only supports two-byte PSM values.

```
typedef UInt16 BtLibL2CapPsmType;
```



### **BtLibLanguageBaseTripletType**

The `BtLibLanguageBaseTripletType` structure represents a language base attribute identifier list attribute. See the “Service Discovery Protocol” chapter of the *Specification of the Bluetooth System* for more information.

```
typedef struct BtLibLanguageBaseTripletType {  
    UInt16 naturalLanguageIdentifier;
```

```
    UInt16 characterEncoding;
    UInt16 baseAttributeID;
} BtLibLanguageBaseTripletType;
```

### Field Descriptions

`naturalLanguageIdentifier`

A UInt16 representing a natural language. See [Language Constants](#) for a set of constants that can be used in this field.

`characterEncoding`

A UInt16 representing a character set encoding. See [Character Encoding Constants](#) for a set of constants that can be used in this field.

`baseAttributeID`

Base attribute identifiers for attributes represented in this language. See [Attribute Identifier Offsets](#) for offsets that are added to this value to get the attribute identifiers for specific attributes represented in this language.

### Language Constants

These constants are used in the `naturalLanguageIdentifier` field of the `BtLibLanguageBaseTripletType` and are defined in the ISO 639:1988 specification.

- `btLibLangAfar`
- `btLibLangAbkikhazian`
- `btLibLangAfrikaans`
- `btLibLangAmharic`
- `btLibLangArabic`
- `btLibLangAssamese`
- `btLibLangAymara`
- `btLibLangAzerbaijani`
- `btLibLangBashkir`
- `btLibLangByelorussian`
- `btLibLangBulgarian`
- `btLibLangBihari`
- `btLibLangBislama`

## **Bluetooth Library: Sockets and Service Discovery**

### *Socket-Related Data Structures*

---

btLibLangBengali  
btLibLangTibetan  
btLibLangBreton  
btLibLangCatalan  
btLibLangCorsican  
btLibLangCzech  
btLibLangWelsh  
btLibLangDanish  
btLibLangGerman  
btLibLangBhutani  
btLibLangGreek  
btLibLangEnglish  
btLibLangEsperanto  
btLibLangSpanish  
btLibLangEstonian  
btLibLangBasque  
btLibLangPersian  
btLibLangFinnish  
btLibLangFiji  
btLibLangFaroese  
btLibLangFrench  
btLibLangFrisian  
btLibLangIrish  
btLibLangScotsGaelic  
btLibLangGalician  
btLibLangGuarani  
btLibLangGujarati  
btLibLangHausa  
btLibLangHindi  
btLibLangCroation  
btLibLangHungarian  
btLibLangArmenian  
btLibLangInterlingua  
btLibLangInterlingue  
btLibLangInupiak  
btLibLangIndonesian  
btLibLangIcelandic  
btLibLangItalian  
btLibLangHebrew  
btLibLangJapanese

btLibLangYiddish  
btLibLangJavanese  
btLibLangGeorgian  
btLibLangKazakh  
btLibLangGreenlandic  
btLibLangCambodian  
btLibLangKannada  
btLibLangKorean  
btLibLangKashmiri  
btLibLangKurdish  
btLibLangKirghiz  
btLibLangLatin  
btLibLangLingala  
btLibLangLaothian  
btLibLangLithuanian  
btLibLangLatvian  
btLibLangMalagasy  
btLibLangMaori  
btLibLangMacedonian  
btLibLangMalayalam  
btLibLangMongolian  
btLibLangMoldavian  
btLibLangMarathi  
btLibLangMalay  
btLibLangMaltese  
btLibLangBurmese  
btLibLangNaura  
btLibLangNepali  
btLibLangDutch  
btLibLangNorwegian  
btLibLangOccitan  
btLibLangOromo  
btLibLangOriya  
btLibLangPunjabi  
btLibLangPolish  
btLibLangPashto  
btLibLangPortuguese  
btLibLangQuechua  
btLibLangRhaeto\_Romance  
btLibLangKirundi

## **Bluetooth Library: Sockets and Service Discovery**

### *Socket-Related Data Structures*

---

btLibLangRomanian  
btLibLangRussian  
btLibLangKinyarwanda  
btLibLangSanskrit  
btLibLangSindhi  
btLibLangSangho  
btLibLangSerbo\_Croatian  
btLibLangSinghalese  
btLibLangSlovak  
btLibLangSlovenian  
btLibLangSamoan  
btLibLangShona  
btLibLangSomali  
btLibLangAlbanian  
btLibLangSerbian  
btLibLangSiswati  
btLibLangSesotho  
btLibLangSundanese  
btLibLangSwedish  
btLibLangSwahili  
btLibLangTamil  
btLibLangTelugu  
btLibLangTajik  
btLibLangThai  
btLibLangTigrinya  
btLibLangTurkmen  
btLibLangTagalog  
btLibLangSetswanna  
btLibLangTonga  
btLibLangTurkish  
btLibLangTsonga  
btLibLangTatar  
btLibLangTwi  
btLibLangUkrainian  
btLibLangUrdu  
btLibLangUzbek  
btLibLangVietnamese  
btLibLangVolapuk  
btLibLangWolof  
btLibLangXhosa

btLibLangYoruba  
btLibLangChinese  
btLibLangZulu

### **Character Encoding Constants**

These constants are used to specify the character encoding used for SDP attributes. More information about these character sets can be found at <http://www.iana.org/assignments/character-sets>.

btLibCharSet\_US\_ASCII  
btLibCharSet\_Adobe\_Standard\_Encoding  
btLibCharSet\_Adobe\_Symbol\_Encoding  
btLibCharSet\_ANSI\_X3\_110\_1983  
btLibCharSet\_ASMO\_449  
btLibCharSet\_Big5  
btLibCharSet\_Big5\_HKSCS  
btLibCharSet\_BS\_4730  
btLibCharSet\_BS\_viewdata  
btLibCharSet\_CSA\_Z243\_4\_1985\_1  
btLibCharSet\_CSA\_Z243\_4\_1985\_2  
btLibCharSet\_CSA\_Z243\_4\_1985\_gr  
btLibCharSet\_CSN\_369103  
btLibCharSet\_DEC\_MCS  
btLibCharSet\_DIN\_66003  
btLibCharSet\_dk\_us  
btLibCharSet\_DS\_2089  
btLibCharSet\_EBCDIC\_AT\_DE  
btLibCharSet\_EBCDIC\_AT\_DE\_A  
btLibCharSet\_EBCDIC\_CA\_FR  
btLibCharSet\_EBCDIC\_DK\_NO  
btLibCharSet\_EBCDIC\_DK\_NO\_A  
btLibCharSet\_EBCDIC\_ES  
btLibCharSet\_EBCDIC\_ES\_A  
btLibCharSet\_EBCDIC\_ES\_S  
btLibCharSet\_EBCDIC\_FI\_SE  
btLibCharSet\_EBCDIC\_FI\_SE\_A  
btLibCharSet\_EBCDIC\_FR  
btLibCharSet\_EBCDIC\_IT  
btLibCharSet\_EBCDIC\_PT

## **Bluetooth Library: Sockets and Service Discovery**

### *Socket-Related Data Structures*

---

```
btLibCharSet_EBCDIC_UK
btLibCharSet_EBCDIC_US
btLibCharSet_ECMA_cyrilllic
btLibCharSet_ES_23
btLibCharSet_ES2
btLibCharSet_EUC_JP
btLibCharSet_EUC_KR
btLibCharSet_Extended_UNIX_Code_Fixed_Width_for
_Japanese
btLibCharSet_GB2312
btLibCharSet_GB_1988_80
btLibCharSet_GB_2312_80
btLibCharSet_GOST_19768_74
btLibCharSet_greek7
btLibCharSet_greek7_old
btLibCharSet_greek_ccitt
btLibCharSet_HP_DeskTop
btLibCharSet_HP_Legal
btLibCharSet_HP_Math8
btLibCharSet_HP_Pi_font
btLibCharSet_hp_roman8
btLibCharSet_HZ_GB_2312
btLibCharSet_IBM037
btLibCharSet_IBM038
btLibCharSet_IBM273
btLibCharSet_IBM274
btLibCharSet_IBM275
btLibCharSet_IBM277
btLibCharSet_IBM278
btLibCharSet_IBM280
btLibCharSet_IBM281
btLibCharSet_IBM284
btLibCharSet_IBM285
btLibCharSet_IBM290
btLibCharSet_IBM297
btLibCharSet_IBM420
btLibCharSet_IBM423
btLibCharSet_IBM424
btLibCharSet_IBM437
btLibCharSet_IBM500
```

btLibCharSet\_IBM775  
btLibCharSet\_IBM850  
btLibCharSet\_IBM851  
btLibCharSet\_IBM852  
btLibCharSet\_IBM855  
btLibCharSet\_IBM857  
btLibCharSet\_IBM860  
btLibCharSet\_IBM861  
btLibCharSet\_IBM862  
btLibCharSet\_IBM863  
btLibCharSet\_IBM864  
btLibCharSet\_IBM865  
btLibCharSet\_IBM866  
btLibCharSet\_IBM868  
btLibCharSet\_IBM869  
btLibCharSet\_IBM870  
btLibCharSet\_IBM871  
btLibCharSet\_IBM880  
btLibCharSet\_IBM891  
btLibCharSet\_IBM903  
btLibCharSet\_IBM904  
btLibCharSet\_IBM905  
btLibCharSet\_IBM918  
btLibCharSet\_IBM1026  
btLibCharSet\_IBM00858  
btLibCharSet\_IBM00924  
btLibCharSet\_IBM01140  
btLibCharSet\_IBM01141  
btLibCharSet\_IBM01142  
btLibCharSet\_IBM01143  
btLibCharSet\_IBM01144  
btLibCharSet\_IBM01145  
btLibCharSet\_IBM01146  
btLibCharSet\_IBM01147  
btLibCharSet\_IBM01148  
btLibCharSet\_IBM01149  
btLibCharSet\_IBM\_Symbols  
btLibCharSet\_IBM\_Thai  
btLibCharSet\_IEC\_P27\_1  
btLibCharSet\_INIS

## **Bluetooth Library: Sockets and Service Discovery**

### *Socket-Related Data Structures*

---

```
btLibCharSet_INIS_8
btLibCharSet_INIS_cyrillic
btLibCharSet_INVARIANT
btLibCharSet_ISO_646_basic_1983
btLibCharSet_ISO_646_irv_1983
btLibCharSet_ISO_2022_CN
btLibCharSet_ISO_2022_CN_EXT
btLibCharSet_ISO_2022_JP
btLibCharSet_ISO_2022_JP_2
btLibCharSet_ISO_2022_KR
btLibCharSet_ISO_2033_1983
btLibCharSet_ISO_5427
btLibCharSet_ISO_5427_1981
btLibCharSet_ISO_5428_1980
btLibCharSet_ISO_6937_2_25
btLibCharSet_ISO_6937_2_add
btLibCharSet_ISO_8859_1
btLibCharSet_ISO_8859_10
btLibCharSet_iso_8859_13
btLibCharSet_iso_8859_14
btLibCharSet_ISO_8859_15
btLibCharSet_ISO_8859_1_Windows_3_0_Latin_1
btLibCharSet_ISO_8859_1_Windows_3_1_Latin_1
btLibCharSet_ISO_8859_2
btLibCharSet_ISO_8859_2_Windows_Latin_2
btLibCharSet_ISO_8859_3
btLibCharSet_ISO_8859_4
btLibCharSet_ISO_8859_5
btLibCharSet_ISO_8859_6
btLibCharSet_ISO_8859_6_E
btLibCharSet_ISO_8859_6_I
btLibCharSet_ISO_8859_7
btLibCharSet_ISO_8859_8
btLibCharSet_ISO_8859_8_E
btLibCharSet_ISO_8859_8_I
btLibCharSet_ISO_8859_9
btLibCharSet_ISO_8859_9_Windows_Latin_5
btLibCharSet_ISO_8859_supp
btLibCharSet_ISO_10367_box
btLibCharSet_ISO_10646_UCS_2
```

btLibCharSet\_ISO\_10646\_UCS\_4  
btLibCharSet\_ISO\_10646\_UCS\_Basic  
btLibCharSet\_ISO\_10646\_Unicode\_Latin1  
btLibCharSet\_ISO\_10646\_UTF\_1  
btLibCharSet\_iso\_ir\_90  
btLibCharSet\_ISO\_Unicode\_IBM\_1261  
btLibCharSet\_ISO\_Unicode\_IBM\_1264  
btLibCharSet\_ISO\_Unicode\_IBM\_1265  
btLibCharSet\_ISO\_Unicode\_IBM\_1268  
btLibCharSet\_ISO\_Unicode\_IBM\_1276  
btLibCharSet\_IT  
btLibCharSet\_JIS\_C6220\_1969\_jp  
btLibCharSet\_JIS\_C6220\_1969\_ro  
btLibCharSet\_JIS\_C6226\_1978  
btLibCharSet\_JIS\_C6226\_1983  
btLibCharSet\_JIS\_C6229\_1984\_a  
btLibCharSet\_JIS\_C6229\_1984\_b  
btLibCharSet\_JIS\_C6229\_1984\_b\_add  
btLibCharSet\_JIS\_C6229\_1984\_hand  
btLibCharSet\_JIS\_C6229\_1984\_hand\_add  
btLibCharSet\_JIS\_C6229\_1984\_kana  
btLibCharSet\_JIS\_Encoding  
btLibCharSet\_JIS\_X0201  
btLibCharSet\_JIS\_X0212\_1990  
btLibCharSet\_JUS\_I\_B1\_002  
btLibCharSet\_JUS\_I\_B1\_003\_mac  
btLibCharSet\_JUS\_I\_B1\_003\_serb  
btLibCharSet\_KOI8\_R  
btLibCharSet\_KOI8\_U  
btLibCharSet\_KSC5636  
btLibCharSet\_KS\_C\_5601\_1987  
btLibCharSet\_latin\_greek  
btLibCharSet\_Latin\_greek\_1  
btLibCharSet\_latin\_lap  
btLibCharSet\_macintosh  
btLibCharSet\_Microsoft\_Publishing  
btLibCharSet\_MNEM  
btLibCharSet\_MNEMONIC  
btLibCharSet\_MSZ\_7795\_3  
btLibCharSet\_NATS\_DANO

## **Bluetooth Library: Sockets and Service Discovery**

### *Socket-Related Data Structures*

---

```
btLibCharSet_NATS_DANO_ADD
btLibCharSet_NATS_SEFI
btLibCharSet_NATS_SEFI_ADD
btLibCharSet_NC_NC00_10_81
btLibCharSet_NF_Z_62_010
btLibCharSet_NF_Z_62_010__1973_
btLibCharSet_NS_4551_1
btLibCharSet_NS_4551_2
btLibCharSet_PC8_Danish_Norwegian
btLibCharSet_PC8_Turkish
btLibCharSet_PT
btLibCharSet_PT2
btLibCharSet_SCSU
btLibCharSet_SEN_850200_B
btLibCharSet_SEN_850200_C
btLibCharSet_Shift_JIS
btLibCharSet_TIS_620
btLibCharSet_T_101_G2
btLibCharSet_T_61_7bit
btLibCharSet_T_61_8bit
btLibCharSet_UNICODE_1_1
btLibCharSet_UNICODE_1_1_UTF_7
btLibCharSet_UNKNOWN_8BIT
btLibCharSet_us_dk
btLibCharSet_UTF_16
btLibCharSet_UTF_16BE
btLibCharSet_UTF_16LE
btLibCharSet_UTF_7
btLibCharSet_UTF_8
btLibCharSet_Ventura_International
btLibCharSet_Ventura_Math
btLibCharSet_Ventura_US
btLibCharSet_videotex_suppl
btLibCharSet_VIQR
btLibCharSet_VISCII
btLibCharSet_Windows_31J
btLibCharSet_windows_1250
btLibCharSet_windows_1251
btLibCharSet_windows_1252
btLibCharSet_windows_1253
```

```
btLibCharSet_windows_1254  
btLibCharSet_windows_1255  
btLibCharSet_windows_1256  
btLibCharSet_windows_1257  
btLibCharSet_windows_1258
```

### Attribute Identifier Offsets

`btLibServiceNameOffset`

The offset from the `baseAttributeID` to get the attribute identifier of the service name.

`btLibServiceDescriptionOffset`

The offset from the `baseAttributeID` to get the attribute identifier of the service description.

`btLibProviderNameOffset`

The offset from the `baseAttributeID` to get the attribute identifier of the provider name.



## BtLibProfileDescriptorListEntryType

The `BtLibProfileDescriptorListEntryType` structure represents an entry in a profile descriptor list attribute. See the “Service Discovery Protocol” chapter of the *Specification of the Bluetooth System* for more information about profile descriptor list attributes.

```
typedef struct  
BtLibProfileDescriptorListEntryType {  
  
    BtLibSdpUuidType profUUID;  
    UInt16 version;  
} BtLibProfileDescriptorListEntryType;
```

### Field Descriptions

`profUUID`      The UUID of the profile.

`version`      The version of the profile.



New

## BtLibProtocolDescriptorListEntryType

The BtLibProtocolDescriptorListEntryType structure represents an entry in a protocol descriptor list attribute. See the “Service Discovery Protocol” chapter of the *Specification of the Bluetooth System* for more information.

```
typedef struct
    BtLibProtocolDescriptorListEntryType {
        BtLibSdpUuidType protoUUID;
        union {
            BtLibL2CapPsmType psm;
            BtLibRfCommServerIdType channel;
        } param;
    } BtLibProtocolDescriptorListEntryType;
```

### Field Descriptions

protoUUID	The UUID of the protocol.
param	A union containing two members: psm and channel. psm is applicable for a L2CAP protocol descriptor and specifies the Protocol and Service Multiplexor. channel is applicable to a RFCOMM protocol descriptor and specifies the server channel.



New

## BtLibRfCommServerIdType

The BtLibRfCommServerIdType type represents a RFCOMM server channel. See the “RFCOMM with TS 07.10” chapter of the *Specification of the Bluetooth System* for more information about server channels.

```
typedef UInt8 BtLibRfCommServerIdType;
```



## BtLibSdpAttributeDataType

The BtLibSdpAttributeDataType union is used to encapsulate an SDP attribute or a list entry in an SDP attribute. The [BtLibSdpServiceRecordGetAttribute](#) function gets an attribute or a list entry and return its contents in a BtLibSdpAttributeDataType. The [BtLibSdpServiceRecordSetAttribute](#) function sets an attribute or list entry according to the contents of a BtLibSdpAttributeDataType. This type supports the universal attributes defined in the *Specification of the Bluetooth System*.

```
typedef union BtLibSdpAttributeDataType {  
    BtLibSdpUuidType serviceClassUuid;  
    UInt32 serviceRecordState;  
    BtLibSdpUuidType serviceIdUuid;  
    BtLibProtocolDescriptorListEntryType  
        protocolDescriptorListEntry;  
    BtLibSdpUuidType browseGroupUuid;  
    BtLibLanguageBaseTripletType  
        languageBaseTripletListEntry;  
    UInt32 timeToLive;  
    UInt8 availability;  
    BtLibProfileDescriptorListEntryType  
        profileDescriptorListEntry;  
    BtLibUrlType documentationUrl;  
    BtLibUrlType clientExecutableUrl;  
    BtLibUrlType iconUrl;  
    BtLibStringType serviceName;  
    BtLibStringType providerName;  
} BtLibSdpAttributeDataType;
```

Note that if you're retrieving a string or a URL using the [BtLibSdpServiceRecordGetAttribute](#) function, you first need to allocate a buffer in addition to this union. This buffer must be large enough to contain the anticipated size of the string or URL. You must also initialize the string pointer and string length fields of the appropriate BtLibAttributeDataType union member. For

## Bluetooth Library: Sockets and Service Discovery

### Socket-Related Data Structures

---

example, if you're retrieving an icon URL, you need to set `iconURL.url` to point to the buffer. You also need to set `iconURL.urlLen` to the length of the buffer.

#### See Also

[BtLibSdpUuidType](#),  
[BtLibProtocolDescriptorListEntryType](#),  
[BtLibProfileDescriptorListEntryType](#),  
[BtLibLanguageBaseTripletType](#), [BtLibUrlType](#),  
[BtLibStringType](#)



## BtLibSdpAttributeIdType

The `BtLibSdpAttributeIdType` type represents a SDP attribute identifier.

```
typedef UInt16 BtLibSdpAttributeIdType;
```

The following constants are defined by the Bluetooth library. They represent the universal attributes in the *Specification of the Bluetooth System*.

### Universal Attribute IDs

```
btLibServiceClassIdList  
btLibServiceRecordState  
btLibServiceId  
btLibProtocolDescriptorList  
btLibBrowseGroupList  
btLibLanguageBaseAttributeIdList  
btLibTimeToLive  
btLibAvailability  
btLibProfileDescriptorList  
btLibDocumentationUrl  
btLibClientExecutableUrl  
btLibIconUrl
```



## New **BtLibSdpRecordHandle**

The **BtLibSdpRecordHandle** type, also called an *SDP memory handle*, provides a memory handle to an *SDP memory record*.

```
typedef MemHandle BtLibSdpRecordHandle;
```

A SDP memory record can have two roles: it can contain a local SDP service record or it can refer to an SDP service record on a remote device. In the latter role, the SDP memory record is said to be *mapped* to a service record on the remote device. The [BtLibSdpServiceRecordMapRemote](#) function performs this mapping.



## New **BtLibSdpRemoteServiceRecordHandle**

The **BtLibSdpRemoteServiceRecordHandle** type represents a SDP service record handle on a remote device as defined in the “Service Discovery Protocol” chapter of the *Specification of the Bluetooth System*. The documentation refers to this type as a *remote service record handle*.

```
typedef UInt32  
BtLibSdpRemoteServiceRecordHandle;
```

Note that this type is different from the [BtLibSdpRecordHandle](#) type, which refers to a memory chunk containing an SDP service record.



## New **BtLibSdpUuidSizeEnum**

The **BtLibSdpUuidSizeEnum** enum specifies the sizes that a UUID can have. See [BtLibSdpUuidType](#) for more information.

```
typedef enum {  
    btLibUuidSize16 = 2,  
    btLibUuidSize32 = 4,  
    btLibUuidSize128 = 16  
} BtLibSdpUuidSizeEnum;
```

## Bluetooth Library: Sockets and Service Discovery

### Socket-Related Data Structures

---

#### Value Descriptions

btLibUuidSize16	16-bit UUID
btLibUuidSize32	32-bit UUID
btLibUuidSize128	Full size 128-bit UUID



#### BtLibSdpUuidType

The `BtLibSdpUuidType` structure represents a Universally Unique Identifier (UUID). A UUID is a 128-bit value that is generated in a manner that guarantees that it is different from every other UUID.

The “Service Discovery Protocol” chapter of the *Specification of the Bluetooth System* reserves a set of UUIDs for common Bluetooth services and protocols. You can specify these with 32 bits—the remaining 96 bits have a fixed value. A subset of these can be specified with 16 bits zero-extended to 32 bits. Therefore you can specify a UUID using 16, 32, or 128 bits.

You generally don’t set this type directly. Instead, you use the [`BtLibSdpUuidInitialize`](#) macro.

```
typedef struct BtLibSdpUuidType {  
    BtLibSdpUuidSizeEnum size;  
    UInt8 UUID[16];  
} BtLibSdpUuidType;
```

#### Field Descriptions

size	The number of bits used to specify the UUID. See <a href="#"><code>BtLibSdpUuidSizeEnum</code></a> .
UUID	The value of the UUID. If you’re setting the value of this field, use the <a href="#"><code>BtLibSdpUuidInitialize</code></a> macro.

### Predefined UUIDs

```
btLibSdpUUID_SC_SERVICE_DISCOVERY_SERVER
btLibSdpUUID_SC_BROWSE_GROUP_DESC
btLibSdpUUID_SC_PUBLIC_BROWSE_GROUP
btLibSdpUUID_SC_SERIAL_PORT
btLibSdpUUID_SC_LAN_ACCESS_PPP
btLibSdpUUID_SC_DIALUP_NETWORKING
btLibSdpUUID_SC_IRMC_SYNC
btLibSdpUUID_SC_OBEX_OBJECT_PUSH
btLibSdpUUID_SC_OBEX_FILE_TRANSFER
btLibSdpUUID_SC_IRMC_SYNC_COMMAND
btLibSdpUUID_SC_HEADSET
btLibSdpUUID_SC_CORDLESS_TELEPHONY
btLibSdpUUID_SC_INTERCOM
btLibSdpUUID_SC_FAX
btLibSdpUUID_SC_HEADSET_AUDIO_GATEWAY
btLibSdpUUID_SC_WAP
btLibSdpUUID_SC_WAP_CLIENT
btLibSdpUUID_SC_PNP_INFO
btLibSdpUUID_SC_GENERIC_NETWORKING
btLibSdpUUID_SC_GENERIC_FILE_TRANSFER
btLibSdpUUID_SC_GENERIC_AUDIO
btLibSdpUUID_SC_GENERIC_TELEPHONY

btLibSdpUUID_PROT_SDP
btLibSdpUUID_PROT_RFCOMM
btLibSdpUUID_PROT_TCS_BIN
btLibSdpUUID_PROT_L2CAP
btLibSdpUUID_PROT_IP
btLibSdpUUID_PROT_UDP
btLibSdpUUID_PROT_TCP
btLibSdpUUID_PROT_TCS_AT
btLibSdpUUID_PROT_OBEX
btLibSdpUUID_PROT_FTP
btLibSdpUUID_PROT_HTTP
btLibSdpUUID_PROT_WSP
```



## BtLibSocketEventType

The BtLibSocketEventType structure contains detailed information regarding a socket callback event. All socket events have some common data. Most socket events have additional data specific to those events. The specific data is stored in a union that is part of the BtLibSocketEvent data structure.

```
typedef struct _BtLibSocketEventType {
    BtLibSocketEventEnum event;
    BtLibSocketRef socket;
    Err status;
    union
    {
        ...
        } eventData;
} BtLibSocketEventType,
*BtLibSocketEventTypePtr;
```

### Field Descriptions

event	BtLibSocketEventEnum enum member that indicates which socket event has occurred. See <a href="#">Socket Callback Events</a> .
socket	Socket associated with the event.
status	Status of the event. The <a href="#">Socket Callback Events</a> section gives more details about how to interpret this field for specific events.
eventData	Data associated with the event. The member of this union that is valid depends on the event. See <a href="#">Socket Callback Events</a> for more information.



## BtLibSocketRef

The BtLibSocketRef type identifies a socket.

```
typedef Int16 BtLibSocketRef;
```



## BtLibStringType

The BtLibStringType structure represents a string in an SDP attribute.

```
typedef struct BtLibStringType {
    Char *str;
    UInt16 strLen;
} BtLibStringType;
```

### Field Descriptions

str	An array of characters representing the string. This array is not null-terminated.
strLen	The length of the string.



## BtLibUrlType

The BtLibUrlType structure represents a uniform resource locator in an SDP attribute.

```
typedef struct BtLibUrlType {
    Char *url;
    UInt16 urlLen;
} BtLibUrlType;
```

### Field Descriptions

url	An array of characters representing the URL. This array is not null-terminated.
urlLen	The length of the string.

## Socket Callback Events

The Bluetooth library socket API supports L2CAP, RFCOMM, and SDP, the upper protocols of the Bluetooth specification. As with the

## **Bluetooth Library: Sockets and Service Discovery**

### *Socket Callback Events*

---

management functions, the socket functions are mostly asynchronous. To signal to the application that socket operations have completed, the Bluetooth library generates a *socket callback events* by calling a callback function.

You specify the callback function when you create a socket. When an event occurs, the callback function is called with two parameters: a pointer to a [BtLibSocketEventType](#) structure and a pointer to a user-defined structure.

The [BtLibSocketEventType](#) structure contains an event field, which indicates the reason the callback is called, a status field, which indicates status information associated with the event, a socket field, which indicates the socket associated with the event, and a union of several structures. The member of the union that is valid depends on the event. The meaning of the events is described in the following sections.

### **btLibSocketEventConnectedInbound**

A remote connection has been accepted because the application has called [BtLibSocketRespondToConnection](#).

For this event, the eventData field contains the following field:

```
BtLibSocketRef newSocket;
```

This [BtLibSocketRef](#) contains the reference to the new socket.

If the remote device requests a L2CAP connection, this event is sent to the L2CAP listener socket with a PSM that matches the PSM of the requested connection. The Bluetooth library creates a new socket that exchanges data with the remote device.

If the remote device requests an RFCOMM connection, this event is sent to the RFCOMM listener socket with a server channel that matches the server channel of the requested connection. The Bluetooth library converts the listener socket into socket that exchanges data with the remote device.

### **btLibSocketEventConnectedOutbound**

An outbound connection initiated by a call to [BtLibSocketConnect](#) has completed. The status field is

`btLibErr.NoError` if the connection has completed successfully. Otherwise, the `status` field indicates why the connection failed.

## **btLibSocketEventConnectRequest**

A remote device has requested a connection.

You must respond to this event with a call to [BtLibSocketRespondToConnection](#).

For this event, the `eventData` field contains the following field:

```
BtLibDeviceAddressType requestingDevice;
```

This [BtLibDeviceAddressType](#) contains the address of the remote device requesting the connection.

If the remote device requests a L2CAP connection, this event is sent to the L2CAP listener socket with a PSM that matches the PSM of the request.

If the remote device requests an RFCOMM connection, this event is sent to the RFCOMM listener socket with a server channel that matches the server channel of the request.

To convert a socket into a listener socket use the [BtLibSocketListen](#) function.

## **btLibSocketEventData**

Data has been received on a socket.

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

```
struct {
    UInt16 dataLen;
    UInt8 *data;
} data;
```

### **Value Descriptions**

`dataLen`              The size, in bytes, of the received data.

`data`                A pointer to the received data.

## **Bluetooth Library: Sockets and Service Discovery**

### *Socket Callback Events*

---

#### **btLibSocketEventDisconnected**

The connection has been lost or one of the devices has disconnected. The socket is now invalid. The status field indicates the reason for the disconnection.

#### **btLibSocketEventSdpServiceRecordHandle**

A request for remote service records matching a list of service classes has completed. The application initiated this request by calling the [BtLibSdpServiceRecordsGetByServiceClass](#) function.

If the status field is `btLibErrNoError`, the SDP operation completed successfully, and the `eventData` field contains valid information. Otherwise the SDP operation failed, and the `status` field indicates the reason for the failure.

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

```
struct {
    UInt16 numSrvRec;
    BtLibSdpRemoteServiceRecordHandle
        *serviceRecordList;
} sdpServiceRecordHandles;
```

#### **Value Descriptions**

`numSrvRec`      Number of remote service record handles in the returned array.

`serviceRecordList`      An array of [BtLibSdpRemoteServiceRecordHandles](#) for the service records matching the service class list.

#### **btLibSocketEventSdpGetAttribute**

An attribute request has completed. The application initiated this request by calling the [BtLibSdpServiceRecordGetAttribute](#) function.

If the status field is `btLibErr.NoError`, the operation completed successfully, and the eventData field contains valid data. Otherwise the operation failed, and the status field indicates the reason for the failure.

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

```
struct {
    BtLibSdpAttributeIdType attributeID;
    BtLibSdpRecordHandle recordH;
    union {
        ...
    } info;
} sdpAttribute;
```

### Value Descriptions

<code>attributeID</code>	The attribute identifier of the attribute.
<code>recordH</code>	A handle identifying the SDP memory record from which the attribute is retrieved.
<code>info</code>	A union containing information specific to the event. See <a href="#">The info Field</a> .

### The info Field

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

```
struct {
    BtLibSdpAttributeDataType *attributeValues;
    UInt16 listNumber;
    UInt16 listEntry;
} data;
```

### Value Descriptions

<code>attributeValues</code>	A <a href="#">BtLibSdpAttributeDataType</a> containing the value of the attribute or list entry.
<code>listNumber</code>	The index of the list in which this list entry appears or 0 if the attribute is not a protocol descriptor list. The index of the first list is 0.

## Bluetooth Library: Sockets and Service Discovery

### Socket Callback Events

---

listEntry

The index of the list entry within the list or 0 if the attribute is not a list. The index of the first entry is 0.

## btLibSocketEventSdpGetStringLen

A string or URL length request has completed. The application initiated this request by calling

[BtLibSdpServiceRecordGetStringOrUrlLength](#).

If the status field is `btLibErrNoError`, the operation completed successfully, and length can be found in the `eventData` field. Otherwise the operation failed, and the `status` field indicates the reason for the failure.

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

```
struct {
    BtLibSdpAttributeIdType attributeID;
    BtLibSdpRecordHandle recordH;
    union {
        ...
    } info;
} sdpAttribute;
```

### Value Descriptions

`attributeID` The attribute identifier of the attribute.

`recordH` A handle identifying the SDP memory record from which the string length is retrieved.

`info` A union containing information specific to the event. See [The info Field](#).

### The info Field

For this event, the `info` field contains the following field:

```
UInt16 strLength;
```

This field contains the length of the string or URL represented by the attribute. Bluetooth strings and URLs are not null-terminated.

## **btLibSocketEventSdpGetNumListEntries**

A number of list entries request has completed. The application initiated this request by calling [BtLibSdpServiceRecordGetNumListEntries](#).

If the status field is btLibErrNoError, the operation completed successfully, and the number of list entries can be found in the eventData field. Otherwise the operation failed, and the status field indicates the reason for the failure.

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

```
struct {
    BtLibSdpAttributeIdType attributeID;
    BtLibSdpRecordHandle recordH;
    union {
        ...
    } info;
} sdpAttribute;
```

### **Value Descriptions**

attributeID	The attribute identifier of the attribute.
recordH	A handle identifying the SDP memory record from which the number of list entries is retrieved.
info	A union containing information specific to the event. See <a href="#">The info Field</a> .

### **The info Field**

For this event, the info field contains the following field:

```
UIInt16 numItems;
```

This field contains the number of entries in the list attribute.

## **btLibSocketEventSdpGetNumLists**

A number of lists request has completed. The application initiated this request by calling [BtLibSdpServiceRecordGetNumLists](#).

## Bluetooth Library: Sockets and Service Discovery

### Socket Callback Events

---

If the status field is `btLibErrNoError`, the operation completed successfully, and the number of lists can be found in the `eventData` field. Otherwise the operation failed, and the `status` field indicates the reason for the failure.

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

```
struct {
    BtLibSdpAttributeIdType attributeID;
    BtLibSdpRecordHandle recordH;
    union {
        ...
    } info;
} sdpAttribute;
```

#### Value Descriptions

<code>attributeID</code>	The attribute identifier of the attribute.
<code>recordH</code>	A handle identifying the SDP memory record from which the number of lists is retrieved.
<code>info</code>	A union containing information specific to the event. See <a href="#">The info Field</a> .

#### The info Field

For this event, the `info` field contains the following field:

```
UInt16 numItems;
```

This field contains the number of lists in the protocol list descriptor attribute.

## btLibSocketEventSdpGetRawAttribute

A get raw attribute request has completed. The application initiated the request by calling

[BtLibSdpServiceRecordGetRawAttribute](#).

If the `status` field is `btLibErrNoError`, the operation completed successfully, and the raw attribute can be found in the `eventData` field. Otherwise the operation failed, and the `status` field indicates the reason for the failure.

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

```
struct {
    BtLibSdpAttributeIdType attributeID;
    BtLibSdpRecordHandle recordH;
    union {
        ...
    } info;
} sdpAttribute;
```

### Value Descriptions

<code>attributeID</code>	The attribute identifier of the attribute.
<code>recordH</code>	A handle identifying the SDP memory record from which the raw attribute is retrieved.
<code>info</code>	A union containing information specific to the event. See <a href="#">The info Field</a> .

### The info Field

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

```
struct {
    UInt16 valSize;
    UInt8 *value;
} rawData;
```

### Value Descriptions

<code>valSize</code>	Number of size, in bytes, of the raw attribute value.
<code>value</code>	Byte array containing the raw attribute value.

## **btLibSocketEventSdpGetRawAttributeSize**

A get raw attribute size request has completed. The application initiated this request by calling

[BtLibSdpServiceRecordGetSizeOfRaw Attribute](#).

If the `status` field is `btLibErrNoError`, the operation completed successfully, and the size of the attribute can be found in the

## **Bluetooth Library: Sockets and Service Discovery**

### *Socket Callback Events*

---

eventData field. Otherwise the operation failed, and the status field indicates the reason for the failure.

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

```
struct {
    BtLibSdpAttributeIdType attributeID;
    BtLibSdpRecordHandle recordH;
    union {
        ...
    } info;
} sdpAttribute;
```

#### **Value Descriptions**

attributeID      The attribute identifier of the attribute.

recordH           A handle identifying the SDP memory record from which the size of the raw attribute is retrieved.

info                A union containing information specific to the event. See [The info Field](#).

#### **The info Field**

For this event, the info field contains the following field:

```
UInt16 valSize;
```

This field contains the size, in bytes, of the raw attribute value.

## **btLibSocketEventSdpGetServerChannelByUuid**

A get server channel request has completed. The application initiated this request by calling

[BtLibSdpGetServerChannelByUuid](#).

If the status field is `btLibErrNoError`, the operation completed successfully, and the server channel can be found in the eventData field. Otherwise, the operation failed, and the status field indicates the reason for the failure.

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

```
struct {
    BtLibSdpRemoteServiceRecordHandle
        remoteHandle;
    union {
        ...
    } param;
} sdpByUuid;
```

### Value Descriptions

`remoteHandle` The handle for the remote SDP service record.

`param` A union containing information that depends on the event. See [The param Field](#).

### The param Field

For this event, the `param` field contains the following field:

```
BtLibRfCommServerIdType channel;
```

This [`BtLibRfCommServerIdType`](#) contains the RFCOMM server channel represented by the SDP service record.

## btLibSocketEventSdpGetPsmByUuid

A get PSM request has completed. The application initiated this request by calling [`BtLibSdpGetPSMByUuid`](#).

If the status field is `btLibErrNoError`, the operation completed successfully, and the server channel can be found in the `eventData` field. Otherwise, the operation failed, and the `status` field indicates the reason for the failure.

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

```
struct {
    BtLibSdpRemoteServiceRecordHandle
        remoteHandle;
    union {
        ...
    } param;
```

## Bluetooth Library: Sockets and Service Discovery

### Socket Callback Events

---

```
} sdpByUuid;
```

#### Value Descriptions

remoteHandle	The handle for the remote SDP service record.
param	A union containing information that depends on the event. See <a href="#">The param Field</a> .

#### The param Field

For this event, the param field contains the following field:

```
BtLibL2CapPsmType psm;
```

This [BtLibL2CapPsmType](#) contains the PSM value of the L2CAP channel represented by the SDP service record.

## btLibSocketEventSendComplete

A send request has completed. The application initiated this request by calling [BtLibSocketSend](#).

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

```
struct {
    UInt16 dataLen;
    UInt8 *data;
} data;
```

#### Value Descriptions

dataLen	The number of bytes of data that were actually sent.
data	Not used. This variable does not contain any valid information.

#### Error Conditions

btLibErrNoError	Success.
btLibErrNoAclLink	No ACL Link.

## Socket Disconnection Error Codes

In addition to the standard error codes that can accompany socket events, the status codes accompanying the [btLibSocketEventConnectedInbound](#), [btLibSocketEventConnectedOutbound](#), and [btLibSocketEventDisconnected](#) events can have the following additional values:

<code>btLibL2DiscConfigOptions</code>	Configuration failed due to an unrecognized configuration option.
<code>btLibL2DiscConfigReject</code>	Configuration was rejected (unknown reason).
<code>btLibL2DiscConfigUnacceptable</code>	Configuration failed due to unacceptable parameters.
<code>btLibL2DiscConnNoResources</code>	The remote device is out of resources.
<code>btLibL2DiscConnPsmUnsupported</code>	The remote device does not support the requested protocol service (PSM).
<code>btLibL2DiscConnSecurityBlock</code>	The remote device's security system denied the connection.
<code>btLibL2DiscLinkDisc</code>	The underlying ACL Link was disconnected.
<code>btLibL2DiscQosViolation</code>	The connection was terminated due to a Quality of Service (QOS) violation.
<code>btLibL2DiscReasonUnknown</code>	Disconnection occurred for an unknown reason.
<code>btLibL2DiscRequestTimeout</code>	An L2CAP request timed out.

## Bluetooth Library: Sockets and Service Discovery

### Socket Functions

---

`btLibL2DiscSecurityBlock`

The local security manager refused the connection attempt.

`btLibL2DiscUserRequest`

Disconnection was requested by either the local or remote device.

## Socket Functions

The Bluetooth library uses sockets to represent L2CAP, RFCOMM, and SDP connections. The functions in this section perform general socket tasks and tasks related to L2CAP and RFCOMM sockets. The functions specific to SDP sockets are in the [Service Discovery Protocol Functions](#) section.



### BtLibSocketAdvanceCredit

---

**Purpose** Advance credit to a given RFCOMM connection socket.

**Declared In** `BtLib.h`

**Prototype** `Err BtLibSocketAdvanceCredit (UInt16 btLibRefNum,  
BtLibSocketRef socket, UInt8 credit)`

**Parameters**

-> <code>btLibRefNum</code>	Reference number for the Bluetooth library.
-> <code>socket</code>	RFCOMM socket reference number.
-> <code>credit</code>	Number credits to add to the total number of credits for this socket. The total number of credits represents the number of packets the remote device can send before data flow stops.

**Result** Returns one of the following values:

`btLibErrNoError`

Success

`btLibErrFailed` Too many credits advanced.

`btLibErrSocket` The specified socket is invalid.  
`btLibErrSocketProtocol`  
The specified socket is not an RFCOMM socket.  
`btLibErrSocketRole`  
The specified socket is not connected.

**Comments** RFCOMM uses a credit based flow control mechanism. For each credit the connection has, one packet of data can be sent. When the credits are spent, data flow stops until you advance more credits using this function.  
Multiple calls to this function have a cumulative effect.



---

## BtLibSocketClose

**Purpose** Close a socket, free associated resources, and kill all associated socket connections.

**Declared In** BtLib.h

**Prototype** Err BtLibSocketClose (UInt16 btLibRefNum,  
BtLibSocketRef socket)

**Parameters** -> `btLibRefNum` Reference number for the Bluetooth library.  
-> `socket` Reference number of socket to close.

**Result** Returns one of the following values:

`btLibErrNoError`  
Success.

`btLibErrNotOpen`  
The referenced Bluetooth library is not open.

`btLibErrSocket`  
The specified socket is invalid.

## Bluetooth Library: Sockets and Service Discovery

### Socket Functions

---

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

**Comments** No callback events are generated when closing a socket.

**See Also** [BtLibSocketCreate](#), [BtLibSocketListen](#),  
[BtLibSocketConnect](#), [BtLibSocketRespondToConnection](#)



## BtLibSocketConnect

**Purpose** Create an outbound L2CAP or RFCOMM connection.

**Declared In** BtLib.h

**Prototype** Err BtLibSocketConnect (UInt16 btLibRefNum,  
BtLibSocketRef socket,  
BtLibSocketConnectInfoType \*connectInfo)

**Parameters** -> `btLibRefNum` Reference number for the Bluetooth library.  
-> `socket` Reference number of socket to connect.  
-> `connectInfo` [BtLibSocketConnectInfoType](#) containing Bluetooth device address and protocol-specific connection information.

**Result** Returns one of the following values:

`btLibErrPending`

The results will be returned through a callback event.

`btLibErrNoAclLink`

An ACL link for the remote device does not exist

`btLibErrNotOpen`

The referenced Bluetooth library is not open.

**btLibErrSocket** The specified socket is invalid.

**btLibErrSocketProtocol**

The protocol of the specified socket is not supported. This function only supports the L2CAP and RFCOMM protocols.

**btLibErrSocketRole**

The specified socket is already connected or listening.

**btLibErrStackNotOpen**

The Bluetooth stack failed to initialize when the library was opened.

**Comments**

If the connection succeeds, the [btLibSocketEventConnectedOutbound](#) event is generated and its status field is set to `btLibErrNoError`. If connection fails, the same event is generated with a non-zero status field, or a [btLibSocketEventDisconnected](#) is generated. In both cases, the status field indicates the reason for the failure.

If the connection succeeds, the [btLibSocketEventData](#) event is generated whenever data is received from the remote device. When the channel disconnects, a [btLibSocketEventDisconnected](#) event is generated.

**BtLibSocketConnectInfoType**

The `BtLibSocketConnectInfoType` structure allows you to specify the address of the remote device and data specific to the protocol of the socket. The protocol-specific data is stored as a union; the member of the union that is valid depends on the protocol.

```
typedef struct BtLibSocketConnectInfoType {
    BtLibDeviceAddressTypePtr remoteDeviceP;
    union {
        ...
    } data;
} BtLibSocketConnectInfoType;
```

## Bluetooth Library: Sockets and Service Discovery

### Socket Functions

---

#### Field Description

remoteDeviceP	A pointer to a <a href="#">BtLibDeviceAddressType</a> that contains the address of the remote device.
data	A union containing protocol-specific information. This union has two members: <a href="#">L2Cap</a> , and <a href="#">RfComm</a> .

#### L2Cap

Use the L2Cap union member if you're setting up a L2CAP socket. This member contains the following structure:

```
struct {
    BtLibL2CapPsmType remotePsm;
    UInt16 minRemoteMtu;
    UInt16 localMtu;
} L2Cap;
```

#### Field Descriptions

remotePsm	A <a href="#">BtLibL2CapPsmType</a> representing the protocol and service multiplexer (PSM) identifier of the protocol to which this socket should connect. This identifier is obtained using the Service Discovery Protocol (SDP).
minRemoteMtu	The minimum MTU, or packet size, that your application can support.
localMtu	The MTU, or packet size, of the local device.

#### RfComm

Use the RfComm union member if you're setting up a RFCOMM socket. This member contains the following structure:

```
struct {
    BtLibRfCommServerIdType remoteService;
    UInt16 maxFrameSize;
    UInt8 advancedCredit;
} RfComm;
```

#### Field Descriptions

remoteService	A <a href="#">BtLibRfCommServerIdType</a> representing the RFCOMM service channel on the remote device to which this socket should connect. This identifier is obtained using the Service Discovery Protocol (SDP).
maxFrameSize	The maximum frame size your application can handle. This value must be between <code>BT_RF_MINFRAMESIZE</code> and <code>BT_RF_MAXFRAMESIZE</code> . If your application can handle any frame size, set this value to <code>BT_RF_DEFAULT_FRAMESIZE</code> .
advancedCredit	An amount of credit the socket advances to the remote device when it successfully connects. Additional credit can be advanced using the <a href="#">BtLibSocketAdvanceCredit</a> function once the connection has been established.

**See Also** [BtLibSocketSend](#), [BtLibSocketClose](#)



---

## BtLibSocketCreate

**Purpose** Create a socket with foreground notification. The Bluetooth library supports a maximum of 16 socket connections.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibSocketCreate (UInt16 btLibRefNum,  
                      BtLibSocketRef *socketRefP,  
                      BtLibSocketProcPtr callbackP, UInt32 refCon,  
                      BtLibProtocolEnum socketProtocol)
```

**Parameters**

-> btLibRefNum	Reference number for the Bluetooth library.
<- socketRefP	Pointer to an allocated <a href="#">BtLibSocketRef</a> that contains the socket value upon return. This pointer must not be NULL.

## Bluetooth Library: Sockets and Service Discovery

### Socket Functions

---

-> callbackP	Callback procedure used to respond to socket events. This value must not be NULL.
-> refCon	Caller-defined data to pass to the callback procedure.
-> socketProtocol	Protocol (L2CAP, RFCOMM, or SDP) to use with this socket.

**Result** Returns one of the following values:

btLibErrNoError	Success.
btLibErrNotOpen	The referenced Bluetooth library is not open.
btLibErrParamError	Either socketRefP or callbackP is NULL.
btLibErrStackNotOpen	The Bluetooth stack failed to initialize when the library was opened.
btLibErrTooMany	The maximum number of sockets allocated for the system has already been reached. The Bluetooth library supports a maximum of 16 socket connections.

**Comments** No callback events are generated when creating a socket.

Before terminating, applications should close all of the sockets that they have created.

**See Also** [BtLibSocketConnect](#), [BtLibSocketListen](#), [BtLibSocketClose](#)



## New **BtLibSocketGetInfo**

**Purpose** Retrieve information for a currently open socket.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibSocketGetInfo (UInt16 btLibRefNum,  
BtLibSocketRef socket,  
BtLibSocketInfoEnum infoType, void *valueP,  
UInt32 valueSize)
```

**Parameters**

-> btLibRefNum	Reference number for the Bluetooth library.
-> socket	Reference number for the socket to query.
-> infoType	Type of information to retrieve. See <a href="#">BtLibSocketInfoEnum</a> .
<- valueP	Buffer into which this function stores the result. You must allocate this buffer.
-> valueSize	Size, in bytes, of the valueP buffer. This size must match the size of the retrieved information.

**Result** Returns one of the following values:

**btLibErrNoError**  
Success.

**btLibErrNotOpen**  
The referenced Bluetooth library is not open.

**btLibErrParamError**  
One or more parameters is invalid. Be sure that  
the valueSize parameter matches the size of  
the information you're retrieving.

## Bluetooth Library: Sockets and Service Discovery

### Socket Functions

---

`btLibErrSdpNotMapped`

The SDP socket has not been mapped to a remote SDP service record. This error occurs when you try to obtain the SDP service record handle before you map socket to a remote service record using [BtLibSdpServiceRecordMapRemote](#).

`btLibErrSocket` The specified socket is invalid or not in use.

`btLibErrSocketRole`

The specified socket is not connected or has the wrong role for the request.

`btlibErrSocketProtocol`

The specified socket has the wrong protocol for the request.

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

#### Comments

#### **BtLibSocketInfoEnum**

The `BtLibSocketInfoEnum` enum allows you to specify which information you want to retrieve using the [BtLibSocketGetInfo](#) function.

```
typedef enum {
    btLibSocketInfo_Protocol = 0,
    btLibSocketInfo_RemoteDeviceAddress,
    btLibSocketInfo_SendPending = 100,
    btLibSocketInfo_MaxTxSize,
    btLibSocketInfo_MaxRxSize,
    btLibSocketInfo_L2CapPsm = 200,
    btLibSocketInfo_L2CapChannel,
    btLibSocketInfo_RfCommServerId = 300,
    btLibSocketInfo_RfCommOutstandingCredits,
    btLibSocketInfo_SdpServiceRecordHandle = 400
} BtLibSocketInfoEnum;
```

**Value Descriptions**

`btLibSocketInfo_L2CapChannel`

`BtLibSocketGetInfo` returns a `BtLibL2CapChannelIDType` that represents the channel identifier for this socket. A `BtLibL2CapChannelIDType` is actually a `UInt16`. This information is valid for L2CAP sockets only. See the “Logical Link Control and Adaptation Protocol Specification” chapter of the *Specification of the Bluetooth System* for more information about channel identifiers.

`btLibSocketInfo_L2CapPsm`

`BtLibSocketGetInfo` returns a `BtLibL2CapPsmType` that represents the Protocol and Service Multiplexer (PSM) this socket is using to route packets. This information is only valid for L2CAP sockets.

`btLibSocketInfo_MaxRxSize`

`BtLibSocketGetInfo` returns a `UInt32` representing the maximum packet size the local device can receive.

`btLibSocketInfo_MaxTxSize`

`BtLibSocketGetInfo` returns a `UInt32` representing the maximum packet size the local device can transmit.

`btLibSocketInfo_Protocol`

`BtLibSocketGetInfo` returns a `BtLibProtocolEnum` representing the socket’s protocol. The members of this enum are `btLibL2CapProtocol`, `btLibRfCommProtocol`, and `btLibSdpProtocol`.

`btLibSocketInfo_RemoteDeviceAddress`

`BtLibSocketGetInfo` returns a `BtLibDeviceAddressType` representing the address of the device at the other end of this socket.

## Bluetooth Library: Sockets and Service Discovery

### Socket Functions

---

`btLibSocketInfo_RfCommServerId`  
BtLibSocketGetInfo returns a  
[BtLibRfCommServerIdType](#) that represents  
the socket's RFCOMM server channel. This  
information is valid for RFCOMM sockets only.

`btLibSocketInfo_RfCommOutstandingCredits`  
BtLibSocketGetInfo returns a UInt16  
containing the number of remaining credits on  
this socket. This information is valid for  
RFCOMM sockets only.

`btLibSocketInfo_SdpServiceRecordHandle`  
BtLibSocketGetInfo returns the  
[BtLibSdpRemoteServiceRecordHandle](#)  
for the service record associated with this  
socket. This information is valid for SDP  
sockets only.

`btLibSocketInfo_SendPending`  
BtLibSocketGetInfo returns a Boolean  
indicating whether a send is currently in  
progress.



## BtLibSocketListen

---

**Purpose** Set up an L2CAP or RFCOMM socket as a listener.

**Declared In** BtLib.h

**Prototype** Err BtLibSocketListen (UInt16 btLibRefNum,  
BtLibSocketRef socket,  
BtLibSocketListenInfoType \*listenInfo)

**Parameters** -> btLibRefNum Reference number for the Bluetooth library.  
-> socket Reference number of the socket.

<-> `listenInfo` Protocol-specific listening information. For more information see [BtLibSocketListenInfoType](#). This parameter must not be NULL.

**Result** Returns one of the following values:

<code>btLibErrNoError</code>	Success. The socket is listening for incoming connections.
<code>btLibErrBusy</code>	The given PSM is in use (L2CAP only)
<code>btLibErrNotOpen</code>	The referenced Bluetooth library is not open.
<code>btLibErrParamError</code>	<code>listenInfo</code> is NULL.
<code>btLibErrSocket</code>	The specified socket is invalid.
<code>btLibErrSocketProtocol</code>	The protocol of the specified socket is not supported. This function only supports the L2CAP and RFCOMM protocols.
<code>btLibErrSocketRole</code>	The specified socket is already listening or connected.
<code>btLibErrStackNotOpen</code>	The Bluetooth stack failed to initialize when the library was opened.
<code>btLibErrTooMany</code>	There are no resources to create a listener socket of this type.

**Comments** A listener socket waits for a remote device to initiate a connection to the local device and then generates a [BtLibSocketEventConnectRequest](#) event to notify the application that it needs to handle the connection attempt. You need to respond to this event with a call to [BtLibSocketRespondToConnection](#) on the listener socket to accept or reject the connection.

## Bluetooth Library: Sockets and Service Discovery

### Socket Functions

---

Under certain circumstances, the `listenInfo` parameter acts as an output as well as an input. See the documentation for `BtLibSocketListenInfoType` that follows.

#### **BtLibSocketListenInfoType**

The `BtLibSocketListenInfoType` structure allows you to specify data specific to the protocol of the listening socket.

```
typedef struct BtLibSocketListenInfoType {
    union {
        ...
    } data;
} BtLibSocketListenInfoType;
```

This data is stored in the `data` field, which is a union consisting of two members: [L2Cap](#), and [RfComm](#). The member of the union that is valid depends on the protocol of the listening socket.

#### **L2Cap**

Use the `L2Cap` union member if you're setting up a L2CAP socket as a listener. This member contains the following structure:

```
struct {
    BtLibL2CapPsmType localPsm;
    UInt16 localMtu;
    UInt16 minRemoteMtu;
} L2Cap;
```

#### Field Descriptions

`localPsm`

A [BtLibL2CapPsmType](#) representing the protocol and service multiplexer (PSM) identifier of the protocol to be used with this socket. You can identify your own protocol provided that its PSM value is odd, is within the range of 0x1001 to 0xFFFF, and has the 9th bit (0x0100) set to zero. These limitations are specified by the *Specification of the Bluetooth System*. If you set this field to `BT_L2CAP_RANDOM_PSM`, the `BtLibSocketListen` function automatically creates a suitable PSM for the channel and returns it in this structure.

localMtu	The maximum transmission unit (MTU), or packet size, of the local device.
minRemoteMtu	The minimum packet size that your application can support.

### **RfComm**

Use the RfComm union member if you're setting up a RFCOMM socket as a listener. This member contains the following structure:

```
struct {
    BtLibRfCommServerIdType serviceID;
    UInt16 maxFrameSize;
    UInt8 advancedCredit;
} RfComm;
```

### **Field Descriptions**

serviceID	A <a href="#">BtLibRfCommServerIdType</a> representing the socket's RFCOMM service channel. It is assigned by RFCOMM and returned in this field when you call <a href="#">BtLibSocketListen</a> .
maxFrameSize	The maximum frame size your application can handle. This value must be between <a href="#">BT_RF_MINFRAMESIZE</a> and <a href="#">BT_RF_MAXFRAMESIZE</a> . If your application can handle any frame size, set this value to <a href="#">BT_RF_DEFAULT_FRAMESIZE</a> .
advancedCredit	An amount of credit the socket advances to the remote device when it successfully connects. Additional credit can be advanced using the <a href="#">BtLibSocketAdvanceCredit</a> function once the connection has been established.

**See Also** [BtLibSocketClose](#)

## Bluetooth Library: Sockets and Service Discovery

### Socket Functions

---



New

## BtLibSocketRespondToConnection

**Purpose** Accept or reject an in-bound connection on a given listener socket.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibSocketRespondToConnection  
(UInt16 btLibRefNum, BtLibSocketRef socket,  
Boolean accept)
```

**Parameters**

-> btLibRefNum	Reference number for the Bluetooth library.
-> socket	Reference number of the listener socket.
-> accept	true to accept the connection; false to reject the connection.

**Result** Returns one of the following values:

btLibErrNoError

Success. This status is returned when accept is false.

btLibErrFailed One or more parameters is invalid.

btLibErrPending

The results will be returned through a callback event.

btLibErrNotOpen

The referenced Bluetooth library is not open.

btLibErrSocket The specified socket is invalid or not in use.

btLibErrSocketProtocol

The protocol of the specified socket is not supported. This function only supports the L2CAP and RFCOMM protocols.

btLibErrSocketRole

The specified socket is not a listener socket.

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

**Comments**

You should call this function when you respond to a [`btLibSocketEventConnectRequest`](#) event delivered to a listener socket.

If the connection succeeds, the [`btLibSocketEventConnectedInbound`](#) event is generated and its status field is set to `btLibErrNoError`. If connection fails, the same event is generated with a non-zero status field, or a [`btLibSocketEventDisconnected`](#) is generated. In both cases, the status field indicates the reason for the failure.

Once the connection succeeds, a [`btLibSocketEventData`](#) callback event is generated whenever data received from the remote device. If the channel disconnects, a [`btLibSocketEventDisconnected`](#) is generated.

RFCOMM listener sockets and L2CAP listener sockets behave differently when you call this function. When you respond to an inbound L2CAP connection, a new L2CAP socket is created to exchange data with the remote device, and the L2CAP listener socket continues to listen for more connections. In other words, a single L2CAP listener socket can “spawn” several L2CAP sockets. This mechanism allows you to create a piconet.

On the other hand, when you respond to an RFCOMM connection, the RFCOMM listener socket becomes a connection socket through which you can exchange data with the remote device. If you want to create another RFCOMM connection, you need to create another listener socket.

**See Also**

[BtLibSocketListen](#), [BtLibSocketSend](#), [BtLibSocketClose](#)

## Bluetooth Library: Sockets and Service Discovery

### Socket Functions

---



New

## BtLibSocketSend

**Purpose** Send data over a connected L2CAP or RFCOMM socket.

**Declared In** BtLib.h

**Prototype** Err BtLibSocketSend (UInt16 btLibRefNum,  
BtLibSocketRef socket, UInt8 \*data,  
UInt32 dataLen)

**Parameters**

-> btLibRefNum	Reference number for the Bluetooth library.
-> socket	Reference number of the transmitting socket.
-> data	Pointer to data to send. If the send returns <code>btLibErrPending</code> , the data buffer contents must remain intact until the <a href="#"><code>btLibSocketEventSendComplete</code></a> event occurs.
-> dataLen	Length of data to send. This value must be less than the Maximum Transmission Unit (MTU) for the socket. The MTU indicates the size of the largest packet that the remote device can receive and is determined when the socket is connected.

**Result** Returns one of the following values:

<code>btLibErrPending</code>	The results will be returned through a callback event.
<code>btLibErrBusy</code>	A send is already in process.
<code>btLibErrNoAclLink</code>	An ACL link for the remote device does not exist
<code>btlibErrNotOpen</code>	The referenced Bluetooth library is not open.

`btLibErrSocket` The specified socket is invalid.

`btLibErrSocketProtocol`

The protocol of the specified socket is not supported by this function. You can only send using the L2CAP and RFCOMM protocols.

`btLibErrSocketRole`

The specified socket is not connected.

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

**Comments**

When the data has been sent successfully, a [`btLibSocketEventSendComplete`](#) callback event is generated and its status field is set to `btLibErrNoError`. If the data is not sent successfully, the same callback event is generated with a non-zero status field.

Note that there can be only one send in progress at a time per socket. You must wait for the [`btLibSocketEventSendComplete`](#) event before sending another packet.

**See Also** [BtLibSocketClose](#)

## Service Discovery Protocol Functions

This section describes functions and macros related to the Bluetooth Service Discovery Protocol (SDP).

 **New**

## BtLibSdpCompareUuids

**Purpose** Compare two UUIDs.

**Declared In** BtLib.h

**Prototype** Err BtLibSdpCompareUuids (UInt16 btLibRefNum,  
BtLibSdpUuidType \*uuid1, BtLibSdpUuidType \*uuid2)

**Parameters** -> btLibRefNum Reference number for the Bluetooth Library.  
-> uuid1 UUID to compare.  
-> uuid2 UUID to compare.

**Result** Returns one of the following values:

btLibErrNoError	UUIDs are the same
btLibErrError	UUIDs are different.
btLibErrNotOpen	The referenced Bluetooth library is not open.
btLibErrParamError	One or both UUIDs are invalid.
btLibErrStackNotOpen	The Bluetooth stack failed to initialize when the library was opened.



## BtLibSdpGetPSMByUuid

**Purpose** Get an available L2CAP PSM using SDP.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibSdpGetPsmByUuid (UInt16 btLibRefNum,  
                           BtLibSocketRef socket,  
                           BtLibDeviceAddressType *remoteDeviceP,  
                           BtLibSdpUuidType *serviceUuidList,  
                           UInt8 uuidListLen)
```

**Parameters**

- > `btLibRefNum` Reference number for the Bluetooth library.

- > `socket` Reference number for an SDP socket.

- > `remoteDeviceP` Device address of a remote device to query.  
This parameter must not be NULL.

- > `serviceUuidList` Array of UUIDs that must match those of the service record. This parameter must not be NULL.

- > `uuidListLen` Length of `serviceUuidList`. A maximum of 12 entries is allowed.

**Result** Returns one of the following values:

`btLibErrPending`

The PSM value will be returned through a callback event.

`btLibErrNotOpen`

The referenced Bluetooth library is not open.

`btLibErrOutOfMemory`

Not enough memory to complete request

`btLibErrParamError`

One or more parameters is invalid.

## Bluetooth Library: Sockets and Service Discovery

### Service Discovery Protocol Functions

---

`btLibErrSocket` The specified socket is invalid or not in use.

`btLibErrSocketRole`

The specified socket is not connected.

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

**Comments** This function returns the L2CAP PSM of the first SDP record on the remote device that contains all the specified UUIDs.

This function generates a `btLibSocketEventSdpGetPsmByUuid` event when the query completes or fails.

**See Also** [BtLibSdpGetServerChannelByUuid](#)



## BtLibSdpGetRawDataElementSize

**Purpose** Macro that returns a constant representing the data element's size.

**Declared In** BtLib.h

**Prototype** BtLibSdpGetRawDataElementSize (header)

**Parameters** -> header First byte of a data element

**Result** A constant representing the size of the data element.

**Comments** The first byte of a SDP data element contains the type and size of the data element.

The result of this macro is one of the following constants:

### Data Element Sizes

`btLibDESD_1BYTE`

A 1-byte element. However, if the data element's type is `btLibDETD_NIL` then the size is 0 bytes.

`btLibDESD_2BYTES`

A 2-byte element.

`btLibDESD_4BYTES`

A 4-byte element.

`btLibDESD_8BYTES`

An 8-byte element.

`btLibDESD_16BYTES`

A 16-byte element.

`btLibDESD_ADD_8BITS`

The element's actual data size, in bytes, is contained in the next 8 bits.

`btLibDESD_ADD_16BITS`

The element's actual data size, in bytes, is contained in the next 16 bits.

`btLibDESD_ADD_32BITS`

The element's actual data size, in bytes, is contained in the next 32 bits.

These size constants are discussed in greater detail in the “Service Discovery Protocol” chapter of the *Specification of the Bluetooth System*.

**See Also**

[BtLibSdpGetRawDataElementType](#),  
[BtLibSdpParseRawDataElement](#),  
[BtLibSdpVerifyRawDataElement](#)



## New **BtLibSdpGetRawDataElementType**

**Purpose** Macro that returns an SDP data element's type.

**Declared In** BtLib.h

**Prototype** BtLibSdpGetRawDataElementType (header)  
    -> header                 The first byte of a data element

**Result** The type of the data element.

**Comments** The first byte of a SDP data element contains the type and size of the data element.

The result of this macro is one of the following constants:

### Data Element Types

btLibDETD\_NIL Nil, the null type  
btLibDETD\_UINT Unsigned Integer.  
btLibDETD\_SINT Signed Integer  
btLibDETD\_UUID UUID, a universally unique identifier  
btLibDETD\_TEXT Text string  
btLibDETD\_BOOL Boolean  
btLibDETD\_SEQ Data element sequence  
btLibDETD\_ALT Data element alternative  
btLibDETD\_URL URL, a uniform resource locator

These types are discussed in greater detail in the “Service Discovery Protocol” chapter of the *Specification of the Bluetooth System*.

**See Also** [BtLibSdpGetRawDataElementSize](#),  
[BtLibSdpParseRawDataElement](#),  
[BtLibSdpVerifyRawDataElement](#)



## BtLibSdpGetServerChannelByUuid

**Purpose** Get an available RFCOMM server channel using SDP.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibSdpGetServerChannelByUuid  
(UInt16 btLibRefNum, BtLibSocketRef socket,  
BtLibDeviceAddressType *remoteDeviceP,  
BtLibSdpUuidType *serviceUuidList,  
UInt8 uuidListLen)
```

**Parameters**

- > btLibRefNum Reference number for the Bluetooth library.

- > socket Reference number for an SDP socket.

- > remoteDeviceP Device address of a remote device to query.  
This parameter must not be NULL.

- > serviceUuidList Array of UUIDs that must match those of the service record. This parameter must not be NULL.

- > uuidListLen Length of serviceUuidList. A maximum of 12 entries is allowed.

**Result** Returns one of the following values:

**btLibErrPending**

The server channel will be returned through a callback event.

**btLibErrNotOpen**

The referenced Bluetooth library is not open.

**btLibErrOutOfMemory**

Not enough memory to complete request

**btLibErrParamError**

One or more parameters is invalid.

## Bluetooth Library: Sockets and Service Discovery

### Service Discovery Protocol Functions

---

`btLibErrSocket` The specified socket is invalid or not in use.

`btLibErrSocketRole`

The specified socket is not connected.

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

<b>Comments</b>	This function returns the RFCOMM server channel number of the first SDP record on the remote device that contains all the specified UIDs.  This function generates a <a href="#"><u>btLibSocketEventSdpGetServerChannelBy_Uuid</u></a> event when the query completes or fails.
<b>See Also</b>	<a href="#"><u>BtLibSdpGetPSMByUuid</u></a>



## BtLibSdpParseRawDataElement

---

<b>Purpose</b>	Parse a raw SDP data element to determine where the data field begins and the size of the data field.						
<b>Declared In</b>	<code>BtLib.h</code>						
<b>Prototype</b>	<pre>Err BtLibSdpParseRawDataElement (UInt16 btLibRefNum, const UInt8 *dataElementP, UInt16 *offset, UInt32 *length)</pre>						
<b>Parameters</b>	<table><tr><td>-&gt; <code>btLibRefNum</code></td><td>Reference number for the Bluetooth library.</td></tr><tr><td>-&gt; <code>dataElementP</code></td><td>Pointer to a raw SDP data element.</td></tr><tr><td>&lt;- <code>offset</code></td><td>Offset, in bytes, between <code>dataElementP</code> and the start of the data field.</td></tr></table>	-> <code>btLibRefNum</code>	Reference number for the Bluetooth library.	-> <code>dataElementP</code>	Pointer to a raw SDP data element.	<- <code>offset</code>	Offset, in bytes, between <code>dataElementP</code> and the start of the data field.
-> <code>btLibRefNum</code>	Reference number for the Bluetooth library.						
-> <code>dataElementP</code>	Pointer to a raw SDP data element.						
<- <code>offset</code>	Offset, in bytes, between <code>dataElementP</code> and the start of the data field.						

<- lengthP      Length, in bytes, of the data field.

**Result** Returns one of the following values:

btLibErrNoError  
Successfully parsed the attribute.

btLibErrNotOpen  
The reference Bluetooth library is not open.

btLibErrParamError  
dataElementP, offset, or length is NULL.

btLibErrStackNotOpen  
The Bluetooth stack failed to initialize when the library was opened.

**Comments** A data element has three fields. The first field, called the *header field*, identifies the type of value stored in the data element and the size of the element. The second field, called the *size field*, contains more information about the size of the data if it's not completely specified by the header. Otherwise the size field is omitted. The third field, called the *data field*, contains the data element's actual value.

The offset this function returns is the offset between the start of the data element and the data field. The size this function returns is the the size of the data field. Note that the sum of the offset and the size is the size of the data element.

This function is especially useful for iterating through entries in a list attribute.

The *Specification of the Bluetooth System* has more information about the structure of a data element.

**See Also** [BtLibSdpVerifyRawDataElement](#),  
[BtLibSdpGetRawDataElementType](#),  
[BtLibSdpGetRawDataElementSize](#)



## New **BtLibSdpServiceRecordCreate**

**Purpose** Allocate a memory chunk that represents an SDP service record.

**Declared In** BtLib.h

**Prototype** Err BtLibSdpServiceRecordCreate  
(UInt16 btLibRefNum,  
BtLibSdpRecordHandle \*recordH)

**Parameters** -> btLibRefNum Reference number for the Bluetooth library.  
<- recordH SDP memory handle for the new SDP memory record.

**Result** Returns one of the following values:

btLibErrNoError  
Success.

btLibErrNotOpen  
The referenced Bluetooth library is not open.

btLibErrOutOfMemory  
Not enough memory to allocate the memory chunk.

btLibErrParamError  
recordH is NULL.

btLibErrStackNotOpen  
The Bluetooth stack failed to initialize when the library was opened.

**See Also** [BtLibSdpServiceRecordDestroy](#),  
[BtLibSdpServiceRecordStartAdvertising](#),  
[BtLibSdpServiceRecordStopAdvertising](#)



## BtLibSdpServiceRecordDestroy

**Purpose** Free the memory associated with a SDP memory record.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibSdpServiceRecordDestroy
    (UInt16 btLibRefNum,
     BtLibSdpRecordHandle recordH)
```

**Parameters**

- > btLibRefNum Reference number for the Bluetooth library.
- > recordH SDP memory handle associated with the memory chunk to be freed.

**Result** Returns one of the following values:

`btLibErrNoError`  
Success.

`btLibErrNotOpen`  
The referenced Bluetooth library is not open.

`btLibErrParamError`  
recordH does not refer to an valid SDP memory record.

`btLibErrStackNotOpen`  
The Bluetooth stack failed to initialize when the library was opened.

**Comments** This function stops advertising the record before it frees it.

**See Also** [BtLibSdpServiceRecordCreate](#),  
[BtLibSdpServiceRecordStartAdvertising](#),  
[BtLibSdpServiceRecordStopAdvertising](#)

## Bluetooth Library: Sockets and Service Discovery

### Service Discovery Protocol Functions

---



New

## BtLibSdpServiceRecordGetAttribute

**Purpose** Retrieve the value of a specific attribute in a SDP memory record. If the attribute is a list or a protocol descriptor list (a list of lists), this function retrieves the value of a specific list entry.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibSdpServiceRecordGetAttribute
    (UInt16 btLibRefNum,
     BtLibSdpRecordHandle recordH,
     BtLibSdpAttributeIdType attributeID,
     BtLibSdpAttributeDataType *attributeValues,
     UInt16 listNumber, UInt16 listEntry)
```

**Parameters**

- > `btLibRefNum` Reference number for the Bluetooth library.
- > `recordH` Handle identifying the SDP memory record.
- > `attributeID` Attribute identifier of the attribute to retrieve.
- <- `attributeValues`  
Buffer into which this function stores the attribute's value. You must allocate this buffer. This pointer must not be NULL.
- > `listNumber` List to query if the attribute is a protocol descriptor list. Otherwise this parameter is ignored.
- > `listEntry` Item to get in the list if the attribute is a list attribute. Otherwise this parameter is ignored.

**Result** Returns one of the following values:

`btLibErrNoError`  
Success.

`btLibErrPending`  
The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.

`btLibErrBusy`

The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.

`btLibErrInProgress`

A query is already pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

`btLibErrNoAclLink`

An ACL link to the remote device does not exist.

`btLibErrNotOpen`

The referenced Bluetooth library is not open.

`btLibErrOutOfMemory`

Not enough memory to perform the query.

`btLibErrParamError`

`recordH` is an invalid handle or `attributeValues` is NULL.

`btLibErrSdpAttributeNotSet`

The specified attribute does not exist in the specified service record.

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

**Comments**

If the specified SDP memory record refers to a service record on a remote device, this function generates a [btLibSocketEventSdpGetAttribute](#) event when the result is available or the query fails. In this case, the buffer to which `attributeValues` points must not be freed before the event occurs; making the buffer global ensures that it remains over the duration of the SDP query.

If you are retrieving a string or a URL, you need to allocate additional space. See the documentation for [BtLibSdpAttributeDataType](#) for more information.

## Bluetooth Library: Sockets and Service Discovery

### Service Discovery Protocol Functions

---

This function supports the universal attributes defined in “Service Discovery Protocol” chapter of the *Specification of the Bluetooth System*.

**See Also** [BtLibSdpServiceRecordSetAttribute](#),

[BtLibSdpServiceRecordMapRemote](#),

[BtLibSdpServiceRecordGetNumListEntries](#),

[BtLibSdpServiceRecordGetNumLists](#),

[BtLibSdpServiceRecordGetStringOrUrlLength](#)



## **BtLibSdpServiceRecordGetNumListEntries**

**S**

**Purpose** Get the number of entries in a list attribute.

**Declared In** BtLib.h

**Prototype** Err BtLibSdpServiceRecordGetNumListEntries  
(UInt16 btLibRefNum,  
BtLibSdpRecordHandle recordH,  
BtLibSdpAttributeIdType attributeID,  
UInt16 listNumber, UInt16 \*numEntries)

**Parameters**

- > btLibRefNum Reference number for the Bluetooth library.
- > recordH Handle identifying the SDP memory record.
- > attributeID Attribute identifier of the attribute whose number of list entries is retrieved.
- > listNumber List to query if the attribute is a ProfileDescriptorListEntry. Otherwise this parameter is ignored.
- <- numEntries Number of entries in the list.

**Result** Returns one of the following values:

btLibErrNoError

Success

**btLibErrPending**

The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.

**btLibErrBusy**

The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.

**btLibErrInProgress**

Another query is pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

**btLibErrNoAclLink**

An ACL link to the remote device does not exist.

**btLibErrNotOpen**

The referenced Bluetooth library is not open.

**btLibErrOutOfMemory**

Not enough memory to perform this query.

**btLibErrParamError**

recordH is an invalid handle or numEntries is NULL.

**btLibErrSdpAttributeNotSet**

The specified attribute does not exist in the specified service record.

**btLibErrStackNotOpen**

The Bluetooth stack failed to open when the library was opened.

**Comments**

This function supports the universal attributes defined in “Service Discovery Protocol” chapter of the *Specification of the Bluetooth System*. Specifically, this function gives valid results for ServiceClassIdList, ProtocolDescriptorList, BrowseGroupList, LanguageBaseAttributeIDList, and ProfileDescriptorList attributes.

If the specified SDP memory record refers to a service record on a remote device, this function generates a

## Bluetooth Library: Sockets and Service Discovery

### Service Discovery Protocol Functions

---

[btLibSocketEventSdpGetNumListEntries](#) event when the result is available or the query fails.

#### See Also

[BtLibSdpServiceRecordGetNumLists](#),  
[BtLibSdpServiceRecordGetAttribute](#),  
[BtLibSdpServiceRecordGetStringOrUrlLength](#),  
[BtLibSdpServiceRecordMapRemote](#)



## BtLibSdpServiceRecordGetNumLists

**Purpose** Get the number of lists in a protocol descriptor list SDP attribute.

**Declared In** BtLib.h

**Prototype** Err BtLibSdpServiceRecordGetNumLists  
(UInt16 btLibRefNum,  
BtLibSdpRecordHandle recordH,  
BtLibSdpAttributeIdType attributeID,  
UInt16 \*numLists)

**Parameters**

- > btLibRefNum Reference number for the Bluetooth library.
- > recordH Handle identifying the SDP memory record.
- > attributeID Attribute identifier of the attribute whose number of lists is retrieved.
- <- numLists Number of lists.

**Result** Returns one of the following values:

btLibErrNoError  
Success.

btLibErrPending

The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.

`btLibErrBusy`

The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.

`btLibErrInProgress`

Another query is pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

`btLibErrNoAclLink`

An ACL link to the remote device does not exist.

`btLibErrNotOpen`

The referenced Bluetooth library is not open.

`btLibErrOutOfMemory`

Not enough memory to perform this query.

`btLibErrParamError`

`recordH` is an invalid handle or `numLists` is `NULL`.

`btLibErrSdpAttributeNotSet`

The specified attribute does not exist in the specified service record.

`btLibErrStackNotOpen`

The Bluetooth stack failed to open when the library was opened.

**Comments**

If the specified SDP memory record refers to a service record on a remote device, this function generates a [btLibSocketEventSdpGetNumLists](#) event when the result is available or the query fails.

**See Also**

[BtLibSdpServiceRecordGetNumListEntries](#),  
[BtLibSdpServiceRecordGetAttribute](#),  
[BtLibSdpServiceRecordGetStringOrUrlLength](#),  
[BtLibSdpServiceRecordMapRemote](#)



New

## BtLibSdpServiceRecordGetRawAttribute

**Purpose** Retrieve the value of an attribute of an SDP memory record. The retrieved attribute is in the format defined in the “Service Discovery Protocol” chapter of the *Specification of the Bluetooth System*.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibSdpServiceRecordGetRawAttribute
    (UInt16 btLibRefNum,
     BtLibSdpRecordHandle recordH,
     BtLibSdpAttributeIdType attributeID,
     UInt8 *value, UInt16 *valSize)
```

**Parameters**

-> btLibRefNum	Reference number for the Bluetooth library.
-> recordH	Handle identifying the SDP memory record.
-> attributeID	Attribute identifier of the attribute to retrieve.
<- value	Buffer into which this function stores the retrieved SDP attribute data. You must allocate this buffer. This pointer must not be NULL.
<-> valSize	Size of the value buffer upon entry. This parameter must not be zero. Upon return, contains the number of bytes retrieved.

**Result** Returns one of the following values:

**btLibErrNoError**  
Success.

**btLibErrPending**  
The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.

**btLibErrBusy**  
The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.

**btLibErrInProgress**

A query is already pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

**btLibErrNoAclLink**

An ACL link to the remote device does not exist.

**btLibErrNotOpen**

The referenced Bluetooth library is not open.

**btLibErrOutOfMemory**

Not enough memory to perform the query.

**btLibErrParamError**

recordH is an invalid handle, value is NULL, valSize is 0, or the size of the attribute value is larger than valSize.

**btLibErrSdpAttributeNotSet**

The specified attribute does not exist in the specified service record.

**btLibErrStackNotOpen**

The Bluetooth stack failed to initialize when the library was opened.

**Comments**

If the specified SDP memory record refers to a service record on a remote device, this function generates a [btLibSocketEventSdpGetRawAttribute](#) event when the result is available or the query fails.

**See Also**

[BtLibSdpServiceRecordSetRawAttribute](#),  
[BtLibSdpServiceRecordGetSizeOfRaw Attribute](#),  
[BtLibSdpServiceRecordMapRemote](#)



**New**

## BtLibSdpServiceRecordGetSizeOfRawAttribute

**Purpose** Return the size, in bytes, of any attribute of an SDP memory record.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibSdpServiceRecordGetSizeOfRawAttribute
    (UInt16 btLibRefNum,
     BtLibSdpRecordHandle recordH,
     BtLibSdpAttributeIDType attributeID,
     UInt16 *size)
```

**Parameters**

-> btLibRefNum	Reference number for the Bluetooth library.
-> recordH	Handle identifying the SDP memory record.
-> attributeID	Attribute identifier of the attribute whose size is retrieved.
<- size	Pointer to a UInt16 to store the size of the attribute. This parameter must not be NULL.

**Result** Returns one of the following values:

btLibErrNoError  
Success.

btLibErrPending

The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.

btLibErrBusy

The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.

**btLibErrInProgress**

A query is already pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

**btLibErrNoAclLink**

An ACL link to the remote device does not exist.

**btLibErrNotOpen**

The referenced Bluetooth library is not open.

**btLibErrOutOfMemory**

Not enough memory to perform the query.

**btLibErrParamError**

recordH is an invalid handle or size is NULL.

**btLibErrSdpAttributeNotSet**

The specified attribute does not exist in the specified service record.

**btLibErrStackNotOpen**

The Bluetooth stack failed to initialize when the library was opened.

**Comments**

If the specified SDP memory record refers to a service record on a remote device, this function generates a [btLibSocketEventSdpGetRawAttributeSize](#) event when the result is available or the query fails.

**See Also**

[BtLibSdpServiceRecordGetRawAttribute](#),  
[BtLibSdpServiceRecordMapRemote](#),  
[BtLibSdpServiceRecordSetRawAttribute](#)

## Bluetooth Library: Sockets and Service Discovery

### *Service Discovery Protocol Functions*

---



New

### **BtLibSdpServiceRecordGetStringOrUrlLength**

gth

**Purpose** Get the length of a string or URL attribute in a SDP memory record.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibSdpServiceRecordGetStringOrUrlLength
    (UInt16 btLibRefNum,
     BtLibSdpRecordHandle recordH,
     BtLibSdpAttributeIdType attributeID,
     UInt16 *length)
```

**Parameters**

-> btLibRefNum	Reference number for the Bluetooth library.
-> recordH	Handle identifying the SDP memory record.
-> attributeID	Attribute identifier of the attribute whose length is retrieved.
<- length	Pointer to a UInt16 where the length of the attribute is stored. This parameter cannot be NULL.

**Result** Returns one of the following values:

btLibErrNoError

Success.

btLibErrPending

The specified SDP memory record refers to a service record on a remote device. The result will be returned through a callback event.

btLibErrBusy

The connection is parked. This error can occur only if the SDP memory record refers to a service record on a remote device.

**btLibErrInProgress**

A query is already pending on this socket. This error can occur only if the SDP memory record refers to a service record on a remote device.

**btLibErrNoAclLink**

An ACL link to the remote device does not exist.

**btLibErrNotOpen**

The referenced Bluetooth library is not open.

**btLibErrOutOfMemory**

Not enough memory to perform the query.

**btLibErrParamError**

The recordH does not refer to a valid handle, length is NULL, or the attribute is not a string or a URL.

**btLibErrSdpAttributeNotSet**

The specified attribute does not exist in the specified SDP record.

**btLibErrStackNotOpen**

The Bluetooth stack failed to initialize when the library was opened.

**Comments**

Bluetooth strings do not include a null terminator.

If the SDP memory record refers to a service record on a remote device, this function generates a

[btLibSocketEventSdpGetStringLen](#) event when the result is available or the query fails.

**See Also**

[BtLibSdpServiceRecordGetAttribute](#),  
[BtLibSdpServiceRecordGetNumListEntries](#),  
[BtLibSdpServiceRecordGetNumLists](#),  
[BtLibSdpServiceRecordMapRemote](#)



## New **BtLibSdpServiceRecordMapRemote**

**Purpose** Configure an SDP memory record so it refers to a service record on a remote device.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibSdpServiceRecordMapRemote
    (UInt16 btLibRefNum, BtLibSocketRef socket,
     BtLibDeviceAddressType *remoteDeviceP,
     BtLibSdpRemoteServiceRecordHandle remoteHandle,
     BtLibSdpRecordHandle recordH)
```

**Parameters**

- > btLibRefNum Reference number for the Bluetooth library.
- > socket Reference number of an SDP socket.
- > remoteDeviceP Device to query.
- > remoteHandle Remote service record handle.
- > recordH SDP memory handle of an empty SDP record.

**Result** Returns one of the following values:

- btLibErrNoError  
The mapping was successful.
- btLibErrNotOpen  
The referenced Bluetooth library is not open.
- btLibErrOutOfMemory  
Not enough memory to perform mapping.
- btLibErrParamError  
recordH is invalid or refers to an invalid memory chunk.
- btLibErrSdpMapped  
The SDP memory record is already mapped to a remote service record.

`btLibErrSocket` The specified socket is invalid or not in use.

`btLibErrSocketProtocol`

The specified socket is not an SDP socket.

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

**Comments**

You must create an SDP memory record using [BtLibSdpServiceRecordCreate](#) before using this function.

Note that this function does not copy the contents of the remote service record to the SDP memory record in local memory.



## **BtLibSdpServiceRecordSetAttribute**

**Purpose**

Set the value of an attribute in an SDP memory record. If the attribute is a list or a protocol descriptor list (a list of lists), this function sets the value of a specific list entry. The SDP memory record must represent a local unadvertised service record.

**Declared In**

`BtLib.h`

**Prototype**

```
Err BtLibSdpServiceRecordSetAttribute  
(UInt16 btLibRefNum,  
BtLibSdpRecordHandle recordH,  
BtLibSdpAttributeIdType attributeID,  
BtLibSdpAttributeDataType *attributeValue,  
UInt16 listNumber, UInt16 listEntry)
```

**Parameters**

-> `btLibRefNum` Reference number for the Bluetooth library.

-> `recordH` Handle of the service record to modify.

-> `attributeID` Attribute identifier of the attribute to set.

-> `attributeValue`

Pointer to the new value for the attribute. This pointer must not be NULL.

## Bluetooth Library: Sockets and Service Discovery

### Service Discovery Protocol Functions

---

-> listNumber	List to modify if the attribute is a protocol descriptor list. Otherwise this parameter is ignored.
-> listEntry	Item to set in the list if the attribute is a list attribute. Otherwise this parameter is ignored.

**Result** Returns one of the following values:

btLibErrNoError	Success.
btLibErrAdvertised	An advertised record was passed in recordH. The record must not be advertised.
btLibErrNotOpen	The referenced Bluetooth library is not open.
btLibErrOutOfMemory	Not enough memory to set the attribute.
btLibErrParamError	recordH is invalid or attributeName is NULL.
btLibErrRemoteRecord	A remote record was passed in recordH. The record must be local.
btLibErrStackNotOpen	The Bluetooth stack failed to initialize when the library was opened.

**Comments** This function only works on SDP memory records that are local and not advertised. You can advertise the record after you finish modifying it.

This function supports the universal attributes defined in the *Specification of the Bluetooth System*.

**See Also**

[BtLibSdpServiceRecordGetAttribute](#),  
[BtLibSdpServiceRecordStartAdvertising](#),  
[BtLibSdpServiceRecordStopAdvertising](#)



## BtLibSdpServiceRecordSetAttributesForSocket

**Purpose** Initialize an SDP memory record so it can represent an existing L2CAP or RFCOMM listener socket as a service.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibSdpServiceRecordSetAttributesForSocket
    (UInt16 btLibRefNum, BtLibSocketRef socket,
     BtLibSdpUuidType *serviceUUIDList,
     UInt8 uuidListLen, const Char *serviceName,
     UInt16 serviceNameLen,
     BtLibSdpRecordHandle recordH)
```

**Parameters**

- > btLibRefNum Reference number for the Bluetooth library.
- > socket Reference number for an RFCOMM or L2CAP socket in listening mode.
- > serviceUUIDList List of UUIDs for the service record.
- > uuidListLen Number of entries in serviceUUIDList. A maximum of 12 entries is allowed.
- > serviceName User-friendly name for the service in English.
- > serviceNameLen Size, in bytes, of serviceName.
- > recordH Handle of the service record to be initialized.

**Result** Returns one of the following values:

`btLibErrNoError`  
Success.

`btLibErrAdvertised`  
The record specified by recordH is being advertised. You must stop advertising the record before you can change it.

## **Bluetooth Library: Sockets and Service Discovery**

### *Service Discovery Protocol Functions*

---

`btLibErrNotOpen`

The referenced Bluetooth library is not open.

`btLibErrOutOfMemory`

Not enough memory to store the contents of the SDP record.

`btLibErrParamError`

`recordH` is not a valid record handle.

`btLibErrRemoteRecord`

A remote record was passed in `recordH`.  
Because the service is local, the record must be local.

`btLibErrSocket` The specified socket is invalid or not in use.

`btLibErrSocketRole`

The specified socket is not a listener socket.

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

#### **Comments**

You must first create an SDP record using [BtLibSdpServiceRecordCreate](#). However, the record must not be advertised. In other words, don't call [BtLibSdpServiceRecordStartAdvertising](#) until after calling this function.

#### **See Also**

[BtLibSdpServiceRecordCreate](#), [BtLibSocketListen](#)



## **BtLibSdpServiceRecordSetRawAttribute**

#### **Purpose**

Set the value for an attribute of a SDP memory record. This function allows you to specify the attribute as an array of bytes in the format defined in the "Service Discovery Protocol" chapter of the

*Specification of the Bluetooth System.* The SDP memory record must represent a local unadvertised service record.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibSdpServiceRecordSetRawAttribute  
(UInt16 btLibRefNum,  
BtLibSdpRecordHandle recordH,  
BtLibSdpAttributeIdType attributeID,  
const UInt8 *value, UInt16 valSize)
```

**Parameters**

-> btLibRefNum	Reference number for the Bluetooth library.
-> recordH	Handle identifying the SDP memory record.
-> attributeID	Attribute identifier of the attribute to set.
-> value	Array of bytes containing SDP attribute data in the format defined in the SDP protocol. This parameter must not be NULL.
-> valSize	Size, in bytes, of value. This parameter must not be 0.

**Result** Returns one of the following values:

**btLibErrNoError**  
Success.

**btLibErrAdvertised**  
recordH is being advertised. The record must not be advertised.

**btLibErrNotOpen**  
The referenced Bluetooth library is not open.

**btLibErrOutOfMemory**  
Not enough memory to set the attribute.

**btLibErrParamError**  
recordH is invalid, value is NULL, or valSize is 0.

## Bluetooth Library: Sockets and Service Discovery

### *Service Discovery Protocol Functions*

---

`btLibErrRemoteRecord`

`recordH` refers to a service record on a remote device. The service record must be local.

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

**Comments** If the service record is being advertised, you must stop advertising it before you modify it.

**See Also** [BtLibSdpServiceRecordGetRawAttribute](#),  
[BtLibSdpServiceRecordSetAttribute](#),  
[BtLibSdpServiceRecordStopAdvertising](#),  
[BtLibSdpServiceRecordStartAdvertising](#)



## BtLibSdpServiceRecordsGetByServiceClass

**Purpose** Get the service record handles corresponding to the service classes advertised on a remote device.

**Declared In** BtLib.h

**Prototype** `Err BtLibSdpServiceRecordsGetByServiceClass  
(UInt16 btLibRefNum, BtLibSocketRef socket,  
BtLibDeviceAddressType *remoteDeviceP,  
BtLibSdpUuidType *uuidList, UInt16 uuidListLen,  
BtLibSdpRemoteServiceRecordHandle *srvRecList,  
UInt16 *numSrvRec)`

**Parameters**

- > `btLibRefNum` Reference number for the Bluetooth library.
- > `socket` Reference number of an SDP socket.
- > `remoteDevice` Remote device to query.

-> <code>uuidList</code>	Array of UUIDs identifying the service classes. This parameter must not be NULL.
-> <code>uuidListLen</code>	Number of elements in the <code>uuidList</code> . You can specify a maximum of 12 UUIDs.
<- <code>srvRecList</code>	Array of service record handles into which this function stores the results of the SDP query. You must allocate this array. This pointer must not be NULL.
<-> <code>numSrvRec</code>	Number of service records allocated in <code>srvRecList</code> . This value is sent to the SDP server so it can limit the number of responses. On return, the actual number of records retrieved.

**Result** Returns one of the following values:

<code>btLibErrPending</code>	The results will be returned through a callback event.
<code>btLibErrBusy</code>	The connection to the remote device is parked.
<code>btLibErrInProgress</code>	A SDP query is already in progress on this socket.
<code>btLibErrNoAclLink</code>	An ACL link to the remote device does not exist.
<code>btLibErrNotOpen</code>	The referenced Bluetooth library is not open.
<code>btLibErrOutOfMemory</code>	Not enough memory to perform the query.
<code>btLibErrParamError</code>	One or more parameters are invalid.
<code>btLibErrSocket</code>	The specified socket is invalid or not in use.
<code>btLibErrSocketProtocol</code>	The specified socket is not an SDP socket.

## Bluetooth Library: Sockets and Service Discovery

### *Service Discovery Protocol Functions*

---

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

<b>Comments</b>	You need to allocate <code>srvRecList</code> , an array of <code>BtLibSdpRemoteServiceRecordHandles</code> large enough to accommodate all of the service record handles corresponding to the specified service classes. Specify the size of the array using the <code>numSrvRec</code> parameter.  This function generates a <code>btLibSocketEventSdpServiceRecordHandle</code> event when the matching service records are available or the query fails.
-----------------	---



## **BtLibSdpServiceRecordStartAdvertising**

<b>Purpose</b>	Make visible an SDP memory record representing a local SDP service record. Remote devices can access visible service records through SDP.
<b>Declared In</b>	<code>BtLib.h</code>
<b>Prototype</b>	<pre>Err BtLibSdpServiceRecordStartAdvertising (UInt16 btLibRefNum, BtLibSdpRecordHandle recordH)</pre>
<b>Parameters</b>	<ul style="list-style-type: none"><li>-&gt; <code>btLibRefNum</code> Reference number for the Bluetooth library.</li><li>-&gt; <code>recordH</code> Handle of the service record to make available to remote devices.</li></ul>
<b>Result</b>	Returns one of the following values:  <code>btLibErrNoError</code> Success  <code>btLibErrNotOpen</code> The referenced Bluetooth library is not open.

<code>btLibErrParamError</code>	<code>recordH</code> is not a valid record handle.
<code>btLibErrRemoteRecord</code>	<code>recordH</code> refers to a remote record. The record must be local.
<code>btLibErrSdpAdvertised</code>	The service record is already accessible by remote devices.
<code>btLibErrStackNotOpen</code>	The Bluetooth stack failed to initialize when the library was opened.

**Comments** You cannot modify an SDP memory record while it is available to remote devices.

**See Also** [BtLibSdpServiceRecordStopAdvertising](#)



## BtLibSdpServiceRecordStopAdvertising

**Purpose** Hide an SDP memory record representing a local SDP service record. Remote devices cannot access hidden service records through SDP.

**Declared In** BtLib.h

**Prototype** Err BtLibSdpServiceRecordStopAdvertising  
(UInt16 btLibRefNum,  
BtLibSdpRecordHandle recordH)

**Parameters** -> `btLibRefNum` Reference number for the Bluetooth library.  
-> `recordH` Handle of the service record to hide.

**Result** Returns one of the following values:

## **Bluetooth Library: Sockets and Service Discovery**

### *Service Discovery Protocol Functions*

---

`btLibErrNoErrorHandler`

Success. The SDP record is no longer available to remote devices.

`btLibErrNotOpen`

The referenced Bluetooth library is not open.

`btLibErrParamError`

`recordH` is not a valid record handle.

`btLibErrRemoteRecord`

`recordH` refers to a remote record. The record must be local.

`btLibErrSdpNotAdvertised`

The service record is already hidden from remote devices.

`btLibErrStackNotOpen`

The Bluetooth stack failed to initialize when the library was opened.

#### **See Also**

[BtLibSdpServiceRecordStartAdvertising](#)



## **BtLibSdpUuidInitialize**

---

**Purpose** Macro that sets the value of a UUID.

**Declared In** `BtLib.h`

**Prototype** `BtLibSdpUuidInitialize (uuid, value, valSize)`

**Parameters** `uuid` [BtLibSdpUuidType](#) to initialize.

`value` Array of bytes representing the UUID. The size of this array depends on `valSize`.

`valSize` [BtLibSdpUuidSizeEnum](#) member specifying the size of the `value` array.

**Result** None.



## BtLibSdpVerifyRawDataElement

**Purpose** Verify that a raw SDP data element is properly formed.

**Declared In** BtLib.h

**Prototype**

```
Err BtLibSdpVerifyRawDataElement  
(UInt16 btLibRefNum, const UInt8 *value,  
UInt16 valSize, UInt8 maxLevel)
```

**Parameters**

-> btLibRefNum	Reference number for the Bluetooth library.
-> value	Raw SDP attribute data.
-> valSize	Size of value, in bytes. The size of the data element must be less than or equal to this parameter, otherwise this function fails.
-> maxLevel	Maximum level of recursion over which this function verifies the data element. Must be at least one.

**Result** Returns one of the following values:

btLibErrNoError	SDP data element is properly formatted.
btLibErrError	SDP data element is not properly formatted.
btLibErrNotOpen	The reference Bluetooth library is not open.
btLibErrParamError	value is NULL.
btLibErrStackNotOpen	The Bluetooth stack failed to initialize when the library was opened.

**Comments** This function checks all size descriptors in the element to ensure that the data element fits into the indicated length. In the case of

## Bluetooth Library: Sockets and Service Discovery

### *Application-Defined Functions*

---

data element sequences or alternates, this function calls itself recursively.

The `maxLevel` parameter specifies the maximum number of times this function calls itself. Limiting the recursion level prevents an infinite loop if the data is bad. `maxLevel` must be large enough to handle the complete data element. For example, to verify a simple data element such as an unsigned integer, `maxLevel` must be at least 1. To verify a data element sequence of UUIDs, `maxLevel` must be at least 2.

**See Also** [BtLibSdpParseRawDataElement](#),  
[BtLibSdpGetRawDataElementType](#),  
[BtLibSdpGetRawDataElementSize](#)

## Application-Defined Functions

This section describes the callback functions that handle socket events. These functions are supplied by the developer and can be named anything.



### **BtLibSocketCallback**

---

**Purpose** Signal the result of a Bluetooth socket event. When the event takes place, this callback function is called.

**Declared In** `BtLibTypes.h`

**Prototype** `void (*BtLibSocketProcPtr)(BtLibSocketEventType *sEvent, UInt32 refCon)`

**Parameters** `-> sEvent`      `BtLibSocketEventType` structure containing the event parameters.

-> refCon      General purpose integer which you can use to hold application-specific information. When you call [BtLibSocketCreate](#) to create the socket, you can specify a value to pass to this parameter.

**Result**      Returns nothing.

**Comments**      The event and status of the event are in the sEvent structure. See [Socket Callback Events](#) for more information.

You must specify this callback function when you create a socket. You do this by calling [BtLibSocketCreate](#).

## **Bluetooth Library: Sockets and Service Discovery**

*Application-Defined Functions*

---

# Cryptography Provider Manager

---

The Cryptography Provider Manager (CPM) is a shared library that acts as a framework for cryptographic services. These services are divided into two levels:

- At the application level, the CPM provides an API that any application can use to perform cryptographic operations.
- The operations themselves are not part of the CPM; instead, they're provided by an **algorithm provider** (or AP). To export its functionality to the CPM, an algorithm provider implements a set of callback functions. Third party developers can create and distribute their own APs. The Palm OS provides a default algorithm provider that's used in the absence of alternatives.

The CPM provides the glue between an application and an AP. In general, this glue is invisible: You don't need to know anything about the APs that are available to use their cryptographic services. This makes it possible for an application to use common cryptographic operations without having to be overly concerned with the exact implementation of these operations. It also relieves the algorithm providers from the duty of providing their own invocation API.

---

**NOTE:** In Palm OS 5, only the application level API is offered; the ability to create and install your own algorithm provider will be supported in a later release.

---

## The Default Provider

In Palm OS 5, there is only one algorithm provider. Implemented by RSA, it provides three cryptographic services:

## Cryptography Provider Manager

### Fundamental CPM Functions

---

- Data encryption and decryption using an RC4 symmetric key stream cipher
- Message digest creation (hashing) using the SHA-1 algorithm.
- Message verification through a combination of the two other operations.

All of the Palm OS 5 cryptography operations are provided by RSA.

One of the features of the CPM library is that the developer doesn't need to know anything about cryptography. The functions are designed such that the default settings will yield satisfying results.

## Fundamental CPM Functions

The fundamental cryptography functions are:

- [CPMLibEncrypt](#) performs a data encryption operation.
- [CPMLibDecrypt](#) performs a data decryption operation.
- [CPMLibHash](#) creates a message digest.
- [CPMLibVerify](#) verifies a message that has a signature and certificate.

These functions operate on blocks of data. For example, [CPMLibEncrypt](#) takes a block of data (and a key), encrypts it, and hands back the encrypted data all in one go.

The CPM also provides sets of functions that let you perform these operations sequentially, allowing you to (for example) initialize an encryption "stream," iteratively feed data to the encryption algorithm, and then "finalize" the stream and get the encrypted data back. These functions are (for encryption) [CPMLibEncryptInit](#), [CPMLibEncryptUpdate](#), and [CPMLibEncryptFinal](#).

## Using the Crypto-Info Structures

The cryptography operations that are currently supported by the CPM library rely on four "crypto-info" data structures:

- The [APKeyInfoStruct](#) describes a symmetric key,

- [APCipherInfoStruct](#) describes an encryption/decryption operation.
- [APHashInfoStruct](#) describes message digest creation.
- [APVerifyInfoStruct](#) describes a message verification operation.

When you perform a cryptographic operation, you'll be asked for one or more of these structures. In most cases you can pass in an "empty" structure and the function will populate it for you. However, it's important that you zero the structure first.

For example, the [CPMLibHash](#) function takes a [APHashInfoStruct](#) as an argument. To use the default hashing operations, simply allocate the structure, zero it, and pass it in:

---

```
APHashInfoType hashinfo;
MemSet ((void *)&hashinfo, sizeof (APHashInfoType), 0);
CPMLibHash( ..., &hashInfo, ...);
```

---

When you're done with a crypto-info structure, you must "release" it:

---

```
CPMLibReleaseHashInfo( ..., &hashInfo );
```

---

## Using the Export Functions

The cryptography structures (key, hash, cipher, verification) can be "exported," or encoded into a form that can be cached. These operations are provided by the [CPMLibExportObjectInfo](#) functions. Purposeful details of exporting (and importing, its functional complement) are given in the individual function descriptions, below. This little section touches on a wrinkle of usage.

When you use an export function, you're asked to supply a buffer that can accommodate the encoded data. This means that you have to know how big the encoded data will be; the export function itself tells you the size through its final (reference) argument. Thus, you have to call the export function twice: Once to get the buffer size, and then again to actually get the encoded data. This is demonstrated below:

## Cryptography Provider Manager

*CPM and AP Constants*

---

```
UInt32 length=0;
UInt8 *data=NULL;
Err error;

/* For demo purposes, we're only interested in the last two
 * arguments (the buffer and the buffer length). First, we
 * set the buffer to NULL, and retrieve the length.
 */
error = CPMLibExport... (... NULL, &length);

/* At this pass, we expect the function to tell us that
 * the buffer is too small. Any other return is treated
 * as an error. Note that the <length> argument is reset to
 * to the required allocation length despite the error
 * return.
*/
if (error != cpmErrBufTooSmall)
    return ...; // or whatever

/* Allocate the buffer (for brevity, the example omits
 * the error check).
*/
data = MemPtrNew(length);

/* Call the export function again to retrieve the
 * encoded data.
*/
error = CPMLibExport... ( ... data, &length);

/* This time we want a 'clean' return. */
if (error != errNone) {
    // handle the error
}
```

---

## CPM and AP Constants



### AP Capability Constants

The AP capability values are bitfield constants that represent the capabilities that an AP supports. The capability constants are OR'd into the `flags` field of the [APPProviderInfoStruct](#) structure.

Declared in `CPMLibCommon.h`, the constants are:

<code>APF_MP</code>	Streaming operations (initialize/update/finalize) are supported ("MP" stands for "multiple part"). See " <a href="#">Fundamental CPM Functions</a> " for details. Note that all APs are expected to support block operations.
<code>APF_HW</code>	The AP's algorithms are implemented in hardware (such as a SmartCard).
<code>APF_KEYGEN</code>	Symmetric key operations (including key info import and export) are supported.
<code>APF_KEYPAIRGEN</code>	Asymmetric key operations (including key info import and export) are supported.
<code>APF_KEYDERIVE</code>	Key derivation is supported.
<code>APF_HASH</code>	Hashing operations (including hash info import and export) are supported.
<code>APF_CIPHER</code>	Message encryption and decryption operations (including cipher info import and export) are supported.
<code>APF_SIGN</code>	Message signing operations (including sign info import and export) are supported.
<code>APF_VERIFY</code>	Message verification operations (including verify info import and export) are supported.

---



New

## Block Encryption Mode Constants

The constants listed below represent the various block encryption modes that may be supported by an AP. You can request a particular mode by setting the [APCipherInfoStruct](#).mode field before passing the structure to the [CPMLibEncrypt](#) or [CPMLibDecrypt](#) function. If you don't specify a mode, the AP will choose one for you, and reset the mode field to the chosen mode.

The APModeEnum data type is used to type the encryption mode constants.

```
typedef UInt32 APModeEnum;
```

Note that encryption modes apply to block operations only. Specifying a mode for a stream encryption (through [CPMLibEncryptInit](#) *et al.*) has no effect.

The constants (and the type) are declared in CPMLibConstants.h. The constants are:

apModeTypeUnspecified = 0	The mode isn't specified; the AP uses its default mode.
apModeTypeNone	The encryption mode doesn't apply.
apModeTypeECB	Electronic codebook mode.
apModeTypeCBC	Cipher block chaining mode.
apModeTypeCBC_CTS	Cipher block chaining with cipher text stealing.
apModeTypeCFB	Cipher feedback mode.
apModeTypeOFB	Output feedback mode.
apModeCounter	Counter mode



## Cipher Algorithm Constants

These constants represent the different cipher algorithms that an AP may support. The algorithm type is encoded in the `type` field of the [APKeyInfoStruct](#) structure. You can request a specific algorithm by setting the value of the `type` field before passing the structure to [CPMLibGenerateKey](#). If you don't care, leave the field zero'd (`apAlgorithmTypeUnspecified`); the function will set the field to tell you which algorithm was used.

In the [APKeyInfoStruct](#) structure, the constants are typed as `APAlgorithmEnum`:

```
typedef UInt32 APAlgorithmEnum
```

The constants (and the type) are defined in `CPMLibCommon.h`. There we see the usual "unspecified" constant:

---

<code>apAlgorithmTypeUnspecified = 0</code>	No algorithm specified; the AP will use its default.
---	--

---

The rest of the constants are divided into groups, below, and listed with very little additional explanation: The constants' names are reasonably self-documenting.

### Block Cipher Algorithms

`apSymmetricTypeDES`  
`apSymmetricTypeRC2`  
`apSymmetricTypeRC4`  
`apSymmetricTypeRC5`  
`apSymmetricTypeRC6`  
`apSymmetricTypeDESX_XDX3` ("Strong" DES)  
`apSymmetricType3DES_EDE2`  
`apSymmetricType3DES_EDE3`  
`apSymmetricTypeIDEA`  
`apSymmetricTypeDiamond2`  
`apSymmetricTypeBlowfish`  
`apSymmetricTypeTEA` (Tiny Encryption Algorithm),  
`apSymmetricTypeSAFER` (Safe and Fast Encryption Routine),

# Cryptography Provider Manager

*CPM and AP Constants*

---

apSymmetricType3WAY  
apSymmetricTypeGOST (USSR Government Standard),  
apSymmetricTypeSHARK  
apSymmetricTypeCAST128, apSymmetricTypeSquare,  
apSymmetricTypeSkipjack

## Stream Ciphers

apSymmetricTypePanama, apSymmetricTypeARC4,  
apSymmetricTypeSEAL, apSymmetricTypeWAKE,  
apSymmetricTypeSapphire, apSymmetricTypeBBS

## AES Block Ciphers

apSymmetricTypeRijndael, apSymmetricTypeCAST256,  
apSymmetricTypeTwofish, apSymmetricTypeMARS,  
apSymmetricTypeSerpent

## Asymmetric Key Ciphers

apAsymmetricTypeRSA, apAsymmetricTypeDSA,  
apAsymmetricTypeElgamal,  
apAsymmetricTypeNR (Nyberg-Rueppel),  
apAsymmetricTypeBlumGoldwasser,  
apAsymmetricTypeRabin,  
apAsymmetricTypeRW (Rabin-Williams),  
apAsymmetricTypeLUC, apAsymmetricTypeLUCELG,  
apAsymmetricTypeECDSA, apAsymmetricTypeECNR,  
apAsymmetricTypeECIES, apAsymmetricTypeECDHC,  
apAsymmetricTypeECMQVC

## Key Agreement Ciphers

apKeyAgreementTypeDH (Diffie-Hellman),  
apKeyAgreementTypeDH2 (Unified Diffie-Hellman),  
apKeyAgreementTypeMQV (Menezes-Qu-Vanstone),  
apKeyAgreementTypeLUCDIF, apKeyAgreementTypeXTRDH



## Export Encoding Constants

Constants that represent different data encoding schemes that are used to convert the crypto-info structs ([APKeyInfoStruct](#), [APHashInfoStruct](#), et al.) into a form that can be cached.

The encoding formats are used by the import and export functions, [CPMLibImportKeyInfo](#), [CPMLibExportKeyInfo](#), [CPMLibImportHashInfo](#), [CPMLibExportHashInfo](#), [CPMLibImportVerifyInfo](#), [CPMLibExportVerifyInfo](#), [CPMLibImportCipherInfo](#), and [CPMLibExportCipherInfo](#).

Declared in `CPMLibCommon.h`, the constants are:

---

<code>IMPORT_EXPORT_TYPE_RAW</code>	The default encoding, as defined by the AP.
<code>IMPORT_EXPORT_TYPE_DER</code>	ASN.1 DER encoding.
<code>IMPORT_EXPORT_TYPE_XML</code>	Standardized XML encoding.

---



## Hashing Algorithm Constants

The hashing constants represent various hashing algorithms. If you want to tell the AP to use a specific algorithm in a hashing operation, you would pass one of these constants as an argument to [CPMLibHash](#) or [CPMLibHashInit](#). If you want the default, use `apHashTypeUnspecified`. The algorithm that's actually used is returned through a [APHashInfoStruct](#) structure.

## Cryptography Provider Manager

*CPM and AP Constants*

---

Declared in CPMLibCommon.h, the constants are:

apHashTypeUnspecified = 0	Unspecified hash algorithm; when generating a message digest, the AP decides which algorithm to use.
apHashTypeNone	Don't hash. Use this constant if you want the AP to suppress hashing in an operation, such as verification, that normally performs it
apHashTypeMD2	Rivest Message Digest 2 (MD2)
apHashTypeMD5	Rivest Message Digest 5 (MD5)
apHashTypeSHA1	Secure Hash Algorithm-160 (SHA-1)
apHashTypeSHA256	SHA 256-bit algorithm
apHashTypeSHA384	SHA 384-bit algorithm
apHashTypeSHA512	SHA 512-bit algorithm
apHashTypeHAVAL	HAVAL one-way algorithm
apHashTypeRIPEMD160	RIPEMD 160-bit
apHashTypeTiger	Tiger algorithm
apHashTypePanama	PANAMA algorithm



## Key Class Constants

A key's "class" specifies whether the key is symmetric or asymmetric and, if the latter, whether it's public or private. The key class constants represent these qualities. Key class is encoded in the `keyclass` field of the [APKeyInfoStruct](#) structure.

In the struct, the constants are typed as `APKeyClassEnum`:

```
typedef UInt32 APKeyClassEnum;
```

The constants (and the type) are defined in `CPMLibCommon.h`. The constants are:

---

<code>apKeyClassUnspecified</code> = 0	The key's class is unspecified.
<code>apKeyClassSymmetric</code>	This is a symmetric key.
<code>apKeyClassPublic</code>	This is the public part of an asymmetric key.
<code>apKeyClassPrivate</code>	This is the private part of an asymmetric key.

---



## Key Usage Constants

---

**NOTE:** The key usage constants are currently unused.

---

The key usage constants describe the different ways that an encryption key can be used. How a key is used is encoded in the usage field of the key's [APKeyInfoStruct](#) structure.

Note that the key usage values are mutually exclusive; you can't OR a set of key usage constants to create a "selectively talented" key. The `apKeyUsageAll` constant is the only "multi-purpose" value currently provided.

The `APKeyUsageEnum` data type is used to type the key usage constants:

```
typedef UInt32 APKeyUsageEnum;
```

The constants (and the type) are defined in `CPMLibConstants.h`. The constants are

---

<code>apKeyUsageUnspecified</code> = 0	The key's usage is unspecified.
<code>apKeyUsageAll</code>	The key can be used in any operation.

## Cryptography Provider Manager

*CPM and AP Constants*

---

apKeyUsageCertificateSigning	The key is intended for certificate signing.
apKeyUsageSigning	The key is intended for message signing operations.
apKeyUsageEncryption	The key is intended for key or data encryption operations.
apKeyUsageKeyEncrypting	The key is intended for key encryption operations
apKeyUsageDataEncrypting	The key is intended for data encryption operations.
apKeyUsageMessageIntegrity	The key is intended for message verification operations.

---



## Plaintext Padding Constants

The plaintext padding constants describe the different ways that plaintext is padded before it's encrypted. The padding type is encoded in the padding field of an [APCipherInfoStruct](#) structure.

The APPaddingEnum data type is used to type the padding constants:

```
typedef UInt32 APPaddingEnum;
```

The constants (and the type) are declared in `CPMLibCommon.h`. The constants are:

---

<code>apPaddingTypeUnspecified</code> = 0	The padding is unspecified; the AP will use the default
<code>apPaddingTypeNone</code>	Specifically request that padding <i>not</i> be applied.
<code>apPaddingTypePKCS1Type1</code>	Public Key Cryptography Standard 1, type 1
<code>apPaddingTypePKCS1Type2</code>	Public Key Cryptography Standard 1, type 2
<code>apPaddingTypePKCS5</code>	Public Key Cryptography Standard 5
<code>apPaddingTypeOAEP</code>	Optimal Asymmetric Encryption Padding
<code>apPaddingTypeSSLv23</code>	Secure Sockets Layer version 23

---

## CPM and AP Structures and Data Types



### APCipherInfoStruct

The `APCipherInfoStruct` encapsulates information about an instance of a data encryption or decryption operation. The structure is populated and returned by the data encryption and decryption functions ([CPMLibEncrypt](#), [CPMLibDecrypt](#), [CPMLibEncryptInit](#), and [CPMLibDecryptInit](#)). You can set some of the fields' values yourself (before calling an encryption/decryption function) to fine-tune the impending operation.

## Cryptography Provider Manager

### CPM and AP Structures and Data Types

---

You're responsible for allocating and freeing the APCipherInfoStructs that you need—the CPM never allocates them for you. The APCipherInfoStructs that you create (and actually use) must be released through [CPMLibReleaseCipherInfo](#) before they're freed.

For more information (including examples) on how to use a crypto-info structure, see "[Using the Crypto-Info Structures](#)."

Declared in CPMLibCommon.h, the structure looks like this:

```
struct APCipherInfoStruct {  
    APProviderContextType providerContext;  
    APAlgorithmEnum type;  
    APPaddingEnum padding;  
    UInt8 *iv;  
    UInt32 ivLength;  
    void *algorithmParams;  
};
```

The fields are:

providerContext	Information about the AP that performed (or is requested to perform) the operation. See <a href="#">APProviderContextStruct</a> .
type	Constant that represents the cipher algorithm that was used or that's requested. See <a href="#">Cipher Algorithm Constants</a>
padding	Constant that represents the plaintext padding scheme used by the algorithm. See <a href="#">Plaintext Padding Constants</a> .
iv	Initialization vector.
ivLength	The length of the initialization vector, in bytes.
algorithmParams	Additional data that's fed to the algorithm, as specified by the AP.



## New APHashInfoStruct

The APHashInfoStruct contains information about an instance of a message digest operation. You allocate the structure yourself, zero its contents, set the fields that you're interested in (if any), and then pass it as an argument to the message digest functions ([CPMLibHash](#), [CPMLibHashInit](#), [CPMLibExportHashInfo](#), and so on). When you're finished with the struct, you pass it to [CPMLibReleaseHashInfo](#).

For more information (including examples) on how to use the structure, see “[Using the Crypto-Info Structures](#).”

Declared in CPMLibCommon.h, the structure looks like this:

```
struct APHashInfoStruct {  
    APProviderContextType providerContext;  
    APHashEnum type;  
    UInt32 length;  
};
```

The fields are:

providerContext

Information about the AP that performed (or is requested to perform) the operation. See [APProviderContextStruct](#).

type

The hashing algorithm that was used to create (or is requested to create) the message digest.

length

The length of the digest.

The APHashInfoStruct is one of the crypto-info structures; it's used as an input/output argument to the message digest functions ([CPMLibHash](#), [CPMLibHashInit](#), [CPMLibExportHashInfo](#), and so on). You allocate the structure yourself; before passing the struct to a function, you *must* zero its contents. When you're finished with the struct, you pass it to [CPMLibReleaseHashInfo](#). For more

information on how to use a crypto-info structure, see “[Using the Crypto-Info Structures](#).”



## APKeyInfoStruct

The APKeyInfoStruct holds information about an encryption key.

```
struct APKeyInfoStruct {  
    APProviderContextType providerContext;  
    APAgorithmEnum type;  
    APKeyUsageEnum usage;  
    APKeyClassEnum keyclass;  
    UInt32 length;  
    UInt32 actualLength;  
    UInt16 exportable;  
    UInt16 ephemeral;  
};
```

providerID	Algorithm provider ID number.
type	A code that identifies the type of algorithm, one of the <a href="#">Cipher Algorithm Constants</a> values.
usage	A code that identifies how the algorithm is used, one of the <a href="#">Key Usage Constants</a> values.
keyLength	Length of the key data, in bytes and padded to the next largest word. (Default is 8.)
keyActualLength	The actual, unpadded length of the key data. (Default is 7).
exportable	Can this key be used in a CPMExportKey() call?. 1 if it can, 0 if it can't.
ephemeral	Is this a “one-shot” key? 1 if it is, 0 if it's permanent..



## New APPProviderContextStruct

The APPProviderContextStruct contains information about an instance of a cryptographic operation. The structure is contained (as an APPProviderContextType value) in the four crypto-info structs, [APKeyInfoStruct](#), [APHashInfoStruct](#), [APCipherInfoStruct](#), and [APVerifyInfoStruct](#).

Declared in CPMLibCommon.h, the structure looks like this:

```
struct APPProviderContextStruct {  
    UInt32 providerID;  
    void *localContext;  
};
```

The fields are

---

providerID	Integer that uniquely identifies the provider that's being used in the operation.
localContext	Provider-specific information about the operation.

---



## New APPProviderInfoStruct

The APPProviderInfoStruct contains information about a specific algorithm provider. The structure is returned (as an APPProviderInfoType) by [CPMLibGetProviderInfo](#).

Declared in CPMLibCommon.h, the structure looks like this:

```
struct APPProviderInfoStruct {  
    char name[32];  
    char other[64];  
    UInt32 flags;  
    UInt8 numAlgorithms;  
    Boolean bHardware;  
};
```

## Cryptography Provider Manager

### CPM and AP Structures and Data Types

---

The fields are

name	The human-readable name of the provider.
other	Additional textual information.
flags	A bitfield that publishes the functionality that this provider supports. See <a href="#">AP Capability Constants</a> for a list of values that this field can combine.
numAlgorithms	A count of the algorithms this provider supplies.
bHardware	true if the provider's algorithms are implemented in hardware; false if in software.



### APVerifyInfoStruct

The APVerifyInfoStruct is used by the verification functions ([CPMLibVerify](#), *et al.*) to verify a message. It contains (primarily) the hash operation and cipher operation information that will be used during verification. It's the caller's responsibility to allocate and embed the structure's APHashInfoStruct and APCipherInfoStruct fields before passing the APVerifyInfoStruct to a verification function.

Any APVerifyInfoStructs that you actually use must be released through [CPMLibReleaseVerifyInfo](#) before it's freed. The embedded structures must also be released through [CPMLibReleaseCipherInfo](#) and [CPMLibReleaseHashInfo](#).

For more information (including examples) on how to use a crypto-info structure, see "[Using the Crypto-Info Structures](#)."

Declared in `CPMLibCommon.h`, the structure looks like this:

```
struct APVerifyInfoStruct {  
    APProviderContextType providerContext;
```

```
    APHashInfoType *hashInfoP;
    APCipherInfoType *cipherInfoP;
}
```

The fields are:

---

providerContext	Information about the AP that performed (or is requested to perform) the operation. See <a href="#">APProviderContextStruct</a> .
hashInfoP	<a href="#">APHashInfoStruct</a> that contains the certificate's hash information.
cipherInfoP	<a href="#">APCipherInfoStruct</a> that contains the certificate's cipher information.

---



## CPMInfoStruct

---

Structure that provides information about the CPM library. It's used by the [CPMLibGetInfo](#) function.

```
typedef struct CPMInfoStruct {
    UInt8 numInstances;
    UInt8 numProviders;
    Boolean defaultProviderPresent;
};
```

The fields are:

---

numInstances	The number of clients (applications) that are talking to this library.
numProviders	The number of algorithm providers that this library knows about.
defaultProviderPresent	Does the library contain a default provider? <code>true</code> if it does, otherwise <code>false</code> .

---

# CPM Functions



## CPMLibDecrypt

**Purpose** Decrypts a block of encrypted data.

**Declared In** CPMLib68kInterface.h, CPMLibARMInterface.h

**Prototype**

```
Err CPMLibDecrypt ( UInt16 libRef,
APKeyInfoType *keyInfo,
APCipherInfoType *cipherInfo, UInt8 *inBuffer,
UInt32 inBufferLength, UInt8 *outBuffer,
UInt32 *outBufferLength )
```

<b>Parameters</b>	-> libRef	(68k only) CPM Library reference number.
	-> keyInfo	<a href="#">APKeyInfoStruct</a> that contains the key that was used to encrypt the data. You obtain the structure by importing the <a href="#">APKeyInfoStruct</a> that was exported by the data encryptor.
	<-> cipherInfo	A pointer to an <a href="#">APCipherInfoStruct</a> that describes the parameters that will be used in the decryption operation. You typically retrieve the structure by importing the <a href="#">APCipherInfoStruct</a> that was exported by the data encryptor. If you're using the decryption defaults provided by the CPM library, you can pass in a freshly allocated (and zero'd) <a href="#">APCipherInfoStruct</a> ; in this case, the structure will be populated by this function to reflect the actual decryption parameters.
	-> inBuffer	A pointer to the (encrypted) data that you want to decrypt.
	-> inBufferLength	Length, in bytes, of inBuffer.

<- `outBuffer` A pointer to the buffer into which the decrypted data will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the decrypted data.

<-> `outBufferLength` You pass in the (allocated) length of `outBuffer`, in bytes. The function resets the argument to the amount of data that was actually copied into `outBuffer`. If the function returns `cpmErrBufTooSmall`, `outBufferLength` is set to the minimum buffer size that's needed to accommodate the decrypted data..

**Result** The function returns `errNone` upon success. For other error codes, see [CPM Error Codes](#).

**Comments** This function performs block decryption. For stream decryption, see [CPMLibDecryptInit](#).  
The `keyInfo` and `cipherInfo` must agree on the algorithm type, as specified in their respective `APAlgorithmEnum` fields.  
If `outBuffer` isn't big enough, the function will fail and `outBufferLength` will return the "correct" output buffer size (i.e. large enough to accommodate the encrypted data). When this happens, simply reallocate the output buffer and call `CPMLibDecrypt` again.



## New CPMLibDecryptFinal

**Purpose** Finalizes a stream decryption operation.

**Declared In** CPMLib68kInterface.h, CPMLibARMInterface.h

**Prototype**

```
Err CPMLibDecryptFinal ( UInt16 libRef,
                         APKeyInfoType *keyInfo,
                         APCipherInfoType *cipherInfo, UInt8 *inBuffer,
                         UInt32 inBufferLength, UInt8 *outBuffer,
                         UInt32 *outBufferLength)
```

<b>Parameters</b>	-> libRef	CPM Library reference number (68k only).
	-> keyInfo	Key used to decrypt the data, as returned by <a href="#">CPMLibGenerateKey</a> or <a href="#">CPMLibImportKeyInfo</a> .
	-> cipherInfo	A pointer to the <a href="#">APCipherInfoStruct</a> that was returned by the stream-initializing <a href="#">CPMLibDecryptInit</a> call.
	-> inBuffer	A pointer to the data that you want to encrypt. If you already supplied all the data through previous <a href="#">CPMLibDecryptUpdate</a> calls, pass NULL.
	-> inBufferLength	The length of inBuffer, in bytes. If inBuffer is NULL, pass 0.
	<- outBuffer	A pointer to the buffer where the decrypted inBuffer data will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the decrypted data.

<-> outBufferLength

You pass in the (allocated) length of outBuffer, in bytes. The function resets the argument to the amount of data that was actually copied into outBuffer. If the function returns cpmErrBufTooSmall, outBufferLength is set to the minimum buffer size that's needed to accommodate the decrypted inBuffer data..

**Result** The function returns errNone upon success. For other error codes, see [CPM Error Codes](#).

**Comments** This function (optionally) decrypts a final buffer of data and then closes the decryption stream that was initialized by [CPMLibDecryptInit](#).

Note that outBuffer contains the decrypted inBuffer data only—it doesn't contain all the data that was decrypted by this stream. It's the callers responsibility to accumulate the data that was decrypted by previous, successive [CPMLibDecryptUpdate](#) calls.



---

## CPMLibDecryptInit

**Purpose** Initializes a stream decryption session.

**Declared In** CPMLib68kInterface.h, CPMLibARMIInterface.h

**Prototype** Err CPMLibDecryptInit ( UInt16 libRef,  
APKeyInfoType \*keyInfo,  
APCipherInfoType \*cipherInfo )

**Parameters** -> libRef CPM Library reference number (68k only).

-> keyInfo [APKeyInfoStruct](#) that contains the key that was used to encrypt the data. You obtain the structure by importing the [APKeyInfoStruct](#) that was exported by the data encryptor.

## Cryptography Provider Manager

### CPM Functions

---

<-> cipherInfo

A pointer to an [APCipherInfoStruct](#) that describes the parameters that will be used in the decryption operation. You typically retrieve the structure by importing the [APCipherInfoStruct](#) that was exported by the data encryptor. If you're using the decryption defaults provided by the CPM library, you can pass in a freshly allocated (and zero'd) [APCipherInfoStruct](#); in this case, the structure will be populated by this function to reflect the actual decryption parameters.

**Result** The function returns errNone upon success. For other error codes, see [CPM Error Codes](#).

**Comments** This function initializes a stream decryption operation. To feed data to the operation, you call [CPMLibDecryptUpdate](#) followed by [CPMLibDecryptFinal](#). The "update" function is optional; the "final" function is mandatory. For block encryption, see [CPMLibDecrypt](#).

The keyInfo and cipherInfo must agree on the algorithm type, as specified in their respective APAAlgorithmEnum fields.



**New**

## CPMLibDecryptUpdate

Feeds data to a stream decryption operation.

**Declared In** CPMLib68kInterface.h, CPMLibARMInterface.h

**Prototype**

```
Err CPMLibDecryptUpdate ( UInt16 libRef,
                           APKeyInfoType *keyInfo,
                           APCipherInfoType *cipherInfo, UInt8 *inBuffer,
                           UInt32 inBufferLength, UInt8 *outBuffer,
                           UInt32 *outBufferLength)
```

**Parameters** see [CPMLibEncryptFinal](#)

**Result** The function returns `errNone` upon success. For other error codes, see [CPM Error Codes](#).

**Comments** This function feeds data into the stream decryption session that was started by [CPMLibDecryptInit](#). You can make any number of `CPMLibDecryptUpdate` calls while the stream is open. When you've finished feeding data into the stream, you call [CPMLibDecryptFinal](#).

This function's arguments, return values, and behavior are nearly identical to [CPMLibDecryptFinal](#) (which see for details). The only difference between them is that this function leaves the stream open, and [CPMLibDecryptFinal](#) closes it.



## New CPMLibEncrypt

**Purpose** Encrypts a block of data.

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype**

```
Err CPMLibEncrypt ( UInt16 libRef,
APKeyInfoType *keyInfo,
APCipherInfoType *cipherInfo, UInt8 *inBuffer,
UInt32 inBufferLength, UInt8 *outBuffer,
UInt32 *outBufferLength)
```

<b>Parameters</b>	-> libRef	CPM Library reference number (68k only).
	-> keyInfo	A pointer to an <a href="#">APKeyInfoStruct</a> that represents the key that will be used to encrypt the data. The allocate the structure, zero it, and then populate it by calling <a href="#">CPMLibGenerateKey</a> or <a href="#">CPMLibImportKeyInfo</a> .
	-> cipherInfo	A pointer to an <a href="#">APCipherInfoStruct</a> that you can use to set the parameters of the encryption operation. If the CPM can't satisfy the requirements you specify in the structure, the operation will fail. If you want to use the default cipher settings, pass in a zero'd structure. When the function returns, the structure will be filled with information describing the operation.
	-> inBuffer	A pointer to the data that you want to encrypt.
	-> inBufferLength	The length of inBuffer, in bytes.
	<- outBuffer	A pointer to the buffer where the encrypted data will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the encrypted data.

<-> `outBufferLength`

You pass in the (allocated) length of `outBuffer`, in bytes. The function resets the argument to the amount of data that was actually copied into `outBuffer`. If the function returns `cpmErrBufTooSmall`, `outBufferLength` is set to the minimum buffer size that's needed to accommodate the encrypted data.

**Result** The function returns `errNone` upon success. For other error codes, see [CPM Error Codes](#).

**Comments** This functions performs block encryption. For stream encryption, see [CPMLibEncryptInit](#).

The `keyInfo` and `cipherInfo` must agree on the algorithm type, as specified in their respective `APAlgorithmEnum` fields.

If `outBuffer` isn't big enough, the function will fail and `outBufferLength` will return the "correct" output buffer size (i.e. large enough to accommodate the encrypted data). When this happens, simply reallocate the output buffer and call `CPMLibEncrypt` again.

After you've encrypted the data, you must export the `keyInfo` and `cipherInfo` structures (see [CPMLibExportKeyInfo](#) and [CPMLibExportCipherInfo](#)) so they can be imported, later, by the data decryptor. Secure storage and transmission of the encryption key is the caller's responsibility.

To decrypt encrypted data, you call [CPMLibDecrypt](#) or [CPMLibDecryptInit](#).



## New CPMLibEncryptFinal

**Purpose** Finalizes a stream encryption operation.

**Declared In** CPMLib68kInterface.h, CPMLibARMInterface.h

**Prototype**

```
Err CPMLibEncryptFinal ( UInt16 libRef,
                         APKeyInfoType *keyInfo,
                         APCipherInfoType *cipherInfo, UInt8 *inBuffer,
                         UInt32 inBufferLength, UInt8 *outBuffer,
                         UInt32 *outBufferLength)
```

<b>Parameters</b>	-> libRef	CPM Library reference number (68k only).
	-> keyInfo	Key used to encrypt the data, as returned by <a href="#">CPMLibGenerateKey</a> or <a href="#">CPMLibImportKeyInfo</a> .
	-> cipherInfo	A pointer to the <a href="#">APCipherInfoStruct</a> that was returned in the stream-initializing <a href="#">CPMLibEncryptInit</a> call.
	-> inBuffer	A pointer to the data that you want to encrypt. If you already supplied all the data through previous <a href="#">CPMLibEncryptUpdate</a> calls, pass NULL.
	-> inBufferLength	The length of inBuffer, in bytes. If inBuffer is NULL, pass 0.
	<- outBuffer	A pointer to the buffer where the encrypted inBuffer data will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate the encrypted inBuffer data.

<-> outBufferLength

You pass in the (allocated) length of outBuffer, in bytes. The function resets the argument to the amount of data that was actually copied into outBuffer. If the function returns cpmErrBufTooSmall, outBufferLength is set to the minimum buffer size that's needed to accommodate the encrypted data.

**Result** The function returns errNone upon success. For other error codes, see [CPM Error Codes](#).

**Comments** This function (optionally) encrypts a final buffer of data and then closes the encryption stream that was initialized by [CPMLibEncryptInit](#).

After you've encrypted the data, you should export the keyInfo and cipherInfo structures (see [CPMLibExportKeyInfo](#) and [CPMLibExportCipherInfo](#)) so they can be imported, later, by the data decryptor. Secure storage and transmission of the encryption key is the caller's responsibility.

Note that outBuffer contains the encrypted inBuffer data only—it doesn't contain all the data that was encrypted by this stream. It's the callers responsibility to accumulate the data that was encrypted by previous, successive [CPMLibEncryptUpdate](#) calls.

To decrypt encrypted data, you call [CPMLibDecrypt](#) or [CPMLibDecryptInit](#).



## New CPMLibEncryptInit

**Purpose** Initializes a stream encryption session.

**Declared In** CPMLib68kInterface.h, CPMLibARMInterface.h

**Prototype** Err CPMLibEncryptInit ( UInt16 libRef,  
APKeyInfoType \*keyInfo,  
APCipherInfoType \*cipherInfo )

**Parameters**

-> libRef	CPM Library reference number (68k only).
-> keyInfo	Key used to encrypt the data, as returned by <a href="#">CPMLibGenerateKey</a> or <a href="#">CPMLibImportKeyInfo</a> .
-> cipherInfo	A pointer to an <a href="#">APCipherInfoStruct</a> that you can use to set the parameters of the encryption operation. If the CPM can't satisfy the requirements you specify in the structure, the operation will fail. If you want to use the default cipher settings, pass in a zero'd structure. When the function returns, the structure will be filled with information describing the operation. The structure is used as a cookie in the subsequent <a href="#">CPMLibEncryptUpdate</a> and/or <a href="#">CPMLibEncryptFinal</a> functions.

**Result** The function returns `errNone` upon success. For other error codes,  
see [CPM Error Codes](#).

**Comments** This function initializes a stream encryption operation. To feed data  
to the operation, you call [CPMLibEncryptUpdate](#) followed by  
[CPMLibEncryptFinal](#). The “update” function is optional; the  
“final” function is mandatory. For block encryption, see  
[CPMLibEncrypt](#).

The `keyInfo` and `cipherInfo` must agree on the algorithm type, as specified in their respective `APAlgorithmEnum` fields.

To decrypt encrypted data, you call [CPMLibDecrypt](#) or [CPMLibDecryptInit](#).



## New **CPMLibEncryptUpdate**

**Purpose** Feeds data to a stream encryption operation.

**Declared In** CPMLib68kInterface.h, CPMLibARMInterface.h

**Prototype** Err CPMLibEncryptUpdate ( UInt16 libRef,  
APKeyInfoType \*keyInfo,  
APCipherInfoType \*cipherInfo, UInt8 \*inBuffer,  
UInt32 inBufferLength, UInt8 \*outBuffer,  
UInt32 \*outBufferLength)

**Parameters** see [CPMLibEncryptFinal](#)

**Result** The function returns `errNone` upon success. For other error codes, see [CPM Error Codes](#).

**Comments** This function feeds data into the stream encryption session that was started by [CPMLibEncryptInit](#). You can make any number of `CPMLibEncryptUpdate` calls while the encryption stream is open. When you've finished feeding data into the stream, you call [CPMLibEncryptFinal](#).

This function's arguments, return values, and behavior are nearly identical to [CPMLibEncryptFinal](#) (which see for details). The only difference between them is that this function leaves the stream open, and [CPMLibEncryptFinal](#) closes it.



## New CPMLibExportCipherInfo

**Purpose** Encodes a cipher into a form that can be cached. To reconstitute an exported cipher, pass it to [CPMLibImportCipherInfo](#).

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype** Err CPMLibExportCipherInfo ( UInt16 libRef,  
APCipherInfoType \*cipherInfo, UInt8 encoding,  
UInt8 \*exportBuffer, UInt32 exportBufferLength )

**Parameters**

-> libRef	CPM Library reference number (68k only).
-> cipherInfo	Structure that represents the cipher that you want to export, as created and returned by <a href="#">CPMLibGenerateKey</a> , or as imported through <a href="#">CPMLibImportKeyInfo</a> .
-> encoding	Constant that specifies the type of encoding. One of IMPORT_EXPORT_RAW, IMPORT_EXPORT_DER, or IMPORT_EXPORT_XML. See <a href="#">Export Encoding Constants</a> for details about these formats.
<-> exportBuffer	Buffer into which the function copies the encoded data. The buffer must be allocated by the caller. Point this argument to NULL if you're using the function to retrieve the size of the encoded data.
<-> exportBufferLength	You pass in the size of exportBuffer in bytes. The function returns (through this argument) the size that's required to accommodate the encoded data.

**Result** The function returns errNone upon success. For other error codes, see [CPM Error Codes](#).

**Comments** You call this function twice: Once to get the size of the required exportBuffer, and then again (after allocating the buffer) to get the encoded buffer. See “[Using the Export Functions](#)” for more information and a free sample.



## New **CPMLibExportHashInfo**

**Purpose** Encodes an [APHashInfoStruct](#) into a form that can be cached. To reconstitute an exported hash info, pass it to [CPMLibImportHashInfo](#).

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype** Err CPMLibExportKeyInfo ( UInt16 libRef,  
APHashInfoType \*hashInfo, UInt8 encoding,  
UInt8 \*exportBuffer, UInt32 exportBufferLength )

**Parameters**

-> libRef	CPM Library reference number (68k only).
-> hashInfo	Structure that you want to export.
-> encoding	Constant that specifies the type of encoding. One of IMPORT_EXPORT_RAW, IMPORT_EXPORT_DER, or IMPORT_EXPORT_XML. See <a href="#">Export Encoding Constants</a> for details about these formats.
<-> exportBuffer	Buffer into which the function copies the encoded data. The buffer must be allocated by the caller. Point this argument to NULL if you’re using the function to retrieve the size of the encoded data (see below)

<-> `exportBufferLength`

You pass in the size of `exportBuffer` in bytes; the function returns (through this argument) the size that's required to accommodate the encoded data.

**Result** The function returns `errNone` upon success. For other error codes, see [CPM Error Codes](#).

**Comments** You call this function twice: Once to get the size of the export buffer, and then again (after allocating the buffer) to retrieve the encoded data. See “[Using the Export Functions](#)” for more information and a free sample.



## CPMLibExportKeyInfo

**Purpose** Encodes a [APKeyInfoStruct](#) into a form that can be cached. To reconstitute an exported key, pass it to [CPMLibImportKeyInfo](#).

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype** `Err CPMLibExportKeyInfo ( UInt16 libRef,  
APKeyInfoType *keyInfo, UInt8 encoding,  
UInt8 *exportBuffer, UInt32 exportBufferLength )`

**Parameters**

-> <code>libRef</code>	CPM Library reference number (68k only).
-> <code>keyInfo</code>	Structure that you want to export.
-> <code>encoding</code>	Constant that specifies the type of encoding. One of <code>IMPORT_EXPORT_RAW</code> , <code>IMPORT_EXPORT_DER</code> , or <code>IMPORT_EXPORT_XML</code> . See <a href="#">Export Encoding Constants</a> for details about these formats.

<-> `exportBuffer`

Buffer into which the function copies the encoded data. The buffer must be allocated by the caller. Point this argument to NULL if you're using the function to retrieve the size of the encoded data (see below)

<-> `exportBufferLength`

You pass in the size of `exportBuffer` in bytes; the function returns (through this argument) the size that's required to accommodate the encoded data.

**Result** The function returns `errNone` upon success. For other error codes, see [CPM Error Codes](#).

**Comments** You call this function twice: Once to get the size of the export buffer, and then again (after allocating the buffer) to retrieve the encoded data. See "[Using the Export Functions](#)" for more information and a free sample.



---

## CPMLibExportVerifyInfo

**Purpose** Encodes a [APVerifyInfoStruct](#) into a form that can be cached. To reconstitute an exported key, pass it to [CPMLibImportVerifyInfo](#).

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype** Err CPMLibExportKeyInfo ( UInt16 libRef,  
APVerifyInfoType \*verifyInfo, UInt8 encoding,  
UInt8 \*exportBuffer, UInt32 exportBufferLength )

**Parameters**

-> libRef	CPM Library reference number (68k only).
-> keyInfo	Structure that you want to export.

## Cryptography Provider Manager

### CPM Functions

---

-> encoding	Constant that specifies the type of encoding. One of IMPORT_EXPORT_RAW, IMPORT_EXPORT_DER, or IMPORT_EXPORT_XML. See <a href="#">Export Encoding Constants</a> for details about these formats.
<-> exportBuffer	Buffer into which the function copies the encoded data. The buffer must be allocated by the caller. Point this argument to NULL if you're using the function to retrieve the size of the encoded data (see below)
<-> exportBufferLength	You pass in the size of exportBuffer in bytes; the function returns (through this argument) the size that's required to accommodate the encoded data.

**Result** The function returns errNone upon success. For other error codes, see [CPM Error Codes](#).

**Comments** You call this function twice: Once to get the size of the export buffer, and then again (after allocating the buffer) to retrieve the encoded data. See “[Using the Export Functions](#)” for more information and a free sample.



## New CPMLibGenerateKey

**Purpose** Generates a new symmetric key.

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype**

```
Err CPMLibGenerateKey ( UInt16 libRef,
                        UInt8 *seedData, UInt32 seedLength,
                        APKeyInfoType *keyInfo )
```

**Parameters**

-> libRef	CPM Library reference number (68k only).
-> seedData	Optional data that's used to seed the key generator. Because PalmOS 5 doesn't currently support key "derivation" (identical key generation based on identical seeds) the seed data is, in essence, a no-op, and can be a pointer to 0. However, you may want to supply (and cache) unique seed data today in anticipation of tomorrow's derivation functionality.
-> seedLength	Length of seedData, in bytes. (Pass 0 if *seedData is 0.)
<-> keyInfo	A pointer to a <a href="#">APKeyInfoStruct</a> that will contain the generated key. The keyInfo structure is allocated and owned by the caller. You can specify the desired provider, key-generation scheme, and so on, by setting the APKeyInfoType fields. Or, to retrieve a default key, zero the structure before you pass it in.

**Result** The function returns errNone upon success. For other error codes, see [CPM Error Codes](#).

**Comments** The [APKeyInfoStruct](#) that's populated by this function can be used in subsequent encryption, decryption, and verification operations.

When you're finished using the [APKeyInfoStruct](#), you must release it through [CPMLibReleaseKeyInfo](#).



## CPMLibGetInfo

**Purpose** Retrieves information about the CPM library.

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype** Err CPMLibLibGetInfo ( UInt16 libRef,  
CPMInfoType \*info)

**Parameters**

-> libRef	CPM Library reference number (68k only).
-> info	Pointer to a <a href="#">CPMInfoStruct</a> that returns the information about the library. See <a href="#">CPMInfoStruct</a> for a description.

**Result** The function returns errNone upon success. For other error codes, see [CPM Error Codes](#).



## CPMLibGetProviderInfo

**Purpose** Retrieves information about a specific provider.

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype** Err CPMLibGetProviderInfo ( UInt16 libRef,  
UInt32 providerID,  
APPProviderInfoType \*providerInfo)

**Parameters**

-> libRef	CPM Library reference number (68k only).
-> providerID	ID number that identifies the provider.

<- providerInfo

Structure into which the function places provider information. The structure is allocated and freed by the caller. The function doesn't clear inapplicable fields on the way out.

**Result** The function returns `errNone` upon success. For other error codes, see [CPM Error Codes](#).



## CPMLibHash

**Purpose** Hashes a block of data.

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype** Err CPMLibHash ( UInt16 libRef, APHashEnum type, APHashInfoType \*hashInfo, UInt8 \*inBuffer, UInt32 inBufferLength, UInt8 \*outBuffer, UInt32 \*outBufferLength )

**Parameters** -> libRef CPM Library reference number (68k only).

-> type A constant that represents the hashing algorithm that will be used to create the message digest. See [Hashing Algorithm Constants](#) for a list of constants. If you want the default algorithm, use `apHashTypeUnspecified`.

-> hashInfo A pointer to an [APHashInfoStruct](#) that you can use to set the parameters of the hashing operation. If the CPM can't satisfy the requirements you specify in the structure, the operation will fail. If you want to use the default settings, pass in a zero'd structure. When the function returns, the structure will be filled with information describing the operation.

## Cryptography Provider Manager

### CPM Functions

---

-> `inBuffer` A pointer to the data that you want to hash.

-> `inBufferLength` The length of `inBuffer`, in bytes. If `inBuffer` is NULL, pass 0.

<- `outBuffer` A pointer to the buffer where the hashed `inBuffer` data will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the hashed data.

<-> `outBufferLength` You pass in the (allocated) length of `outBuffer`, in bytes. The function resets the argument to the amount of data that was actually copied into `outBuffer`. If the function returns `cpmErrBufTooSmall`, `outBufferLength` is set to the minimum buffer size that's needed to accommodate the hashed data..

**Result** The function returns `errNone` upon success. For other error codes, see [CPM Error Codes](#).

**Comments** This function performs a block hash operation. For stream hashing, use [CPMLibHashInit](#).



---

## CPMLibHashFinal

**Purpose** Finalizes a hash session and returns the hashed data.

**Declared In** CPMLib.h

**Prototype**

```
Err CPMLibHashFinal ( UInt16 libRef,
                      APHashInfoType *hashInfo, UInt8 *inBuffer,
                      UInt32 inBufferLength, UInt8 *outBuffer,
                      UInt32 *outBufferLength )
```

**Parameters** -> `libRef` CPM Library reference number (68k only).

-> hashInfo	A pointer to the <a href="#">APHashInfoStruct</a> that was returned by <a href="#">CPMLibHashInit</a> .
-> inBuffer	A pointer to the data that you want to hash. If you already supplied all the data through previous <a href="#">CPMLibHashUpdate</a> calls, pass NULL.
-> inBufferLength	The length of inBuffer, in bytes. If inBuffer is NULL, pass 0.
<- outBuffer	A pointer to the buffer where all the data that has been hashed by this stream will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the hashed data.
<-> outBufferLength	You pass in the (allocated) length of outBuffer, in bytes. The function resets the argument to the amount of data that was actually copied into outBuffer. If the function returns cpmErrBufTooSmall, outBufferLength is set to the minimum buffer size that's needed to accommodate the hashed data.

**Result** The function returns errNone upon success. For other error codes, see [CPM Error Codes](#).

**Comments** This function returns all the data that was hashed by the hash stream, and then closes the stream. It follows an initial call to [CPMLibHashInit](#) and some number of calls to [CPMLibHashUpdate](#).



---

## CPMLibHashInit

**Purpose** Initiates a streaming hash operation.

## Cryptography Provider Manager

### CPM Functions

---

<b>Declared In</b>	CPMLib68KInterface.h, CPMLibArmInterface.h				
<b>Prototype</b>	Err CPMLibHash ( UInt16 libRef, APHashInfoType *hashInfo )				
<b>Parameters</b>	<table><tr><td>-&gt; libRef</td><td>CPM Library reference number (68k only).</td></tr><tr><td>-&gt; hashInfo</td><td>A pointer to an <a href="#">APHashInfoStruct</a> that you can use to set the parameters of the hash operation. If the CPM can't satisfy the requirements you specify in the structure, the operation will fail. If you want to use the default settings, pass in a zero'd structure. When the function returns, the structure will be filled with information describing the operation.</td></tr></table>	-> libRef	CPM Library reference number (68k only).	-> hashInfo	A pointer to an <a href="#">APHashInfoStruct</a> that you can use to set the parameters of the hash operation. If the CPM can't satisfy the requirements you specify in the structure, the operation will fail. If you want to use the default settings, pass in a zero'd structure. When the function returns, the structure will be filled with information describing the operation.
-> libRef	CPM Library reference number (68k only).				
-> hashInfo	A pointer to an <a href="#">APHashInfoStruct</a> that you can use to set the parameters of the hash operation. If the CPM can't satisfy the requirements you specify in the structure, the operation will fail. If you want to use the default settings, pass in a zero'd structure. When the function returns, the structure will be filled with information describing the operation.				
<b>Result</b>	The function returns errNone upon success. For other error codes, see <a href="#">CPM Error Codes</a> .				
<b>Comments</b>	This function initializes a streaming hash operation. To feed data to the stream, you call <a href="#">CPMLibHashUpdate</a> followed by <a href="#">CPMLibHashFinal</a> . The "update" function is optional; the "final" function is mandatory. For a block hash, see <a href="#">CPMLibHash</a> .				



## CPMLibHashUpdate

---

<b>Purpose</b>	Sends data to streaming hash operation.		
<b>Declared In</b>	CPMLib68KInterface.h, CPMLibArmInterface.h		
<b>Prototype</b>	Err CPMLibHashUpdate ( UInt16 libRef, APHashInfoType *hashInfo, UInt8 *inBuffer, UInt32 inBufferLength)		
<b>Parameters</b>	<table><tr><td>-&gt; libRef</td><td>CPM Library reference number (68k only).</td></tr></table>	-> libRef	CPM Library reference number (68k only).
-> libRef	CPM Library reference number (68k only).		

-> hashInfo A pointer to the [APHashInfoStruct](#) that was returned by [CPMLibHashInit](#).  
-> inBuffer A pointer to the data that you want to hash.  
-> inBufferLength The length of inBuffer, in bytes.

**Result** The function returns errNone upon success. For other error codes, see [CPM Error Codes](#).

**Comments** This function feeds data into the streaming hash session that was started by [CPMLibHashInit](#). You can make any number of CPMLibHashUpdate calls while the hash stream is open. When you've finished feeding data into the stream, you call [CPMLibHashFinal](#).

Note that this function doesn't return any hashed data. All the data that's hashed by the stream is returned through the [CPMLibHashFinal](#) call.



---

## CPMLibImportCipherInfo

**Purpose** Imports a previously-exported cipher info structure so that it can be used in subsequent operations.

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype** Err CPMLibImportCipherInfo ( UInt16 libRef,  
UInt8 encoding, UInt8 \*importData,  
UInt32 importDataLength,  
APCipherInfoType \*cipherInfo )

**Parameters** -> libRef CPM Library reference number (68k only).

-> encoding Constant that specifies the encoding type that was used to export the key. One of IMPORT\_EXPORT\_RAW, IMPORT\_EXPORT\_DER, or IMPORT\_EXPORT\_XML. See [Export Encoding Constants](#) for details about these formats.

-> importData The encoded data.

-> importDataLength The length of importData, in bytes.

<- cipherInfo [APCipherInfoStruct](#) that returns the imported key. The structure must be allocated before it's passed in.

**Result** The function returns errNone upon success. For other error codes, see [CPM Error Codes](#).



## CPMLibImportHashInfo

---

**Purpose** Imports a previously-exported hash info structure so that it can be used in subsequent operations.

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype** Err CPMLibImportHashInfo ( UInt16 libRef,  
                  UInt8 encoding, UInt8 \*importData,  
                  UInt32 importDataLength,  
                  APHashInfoType \*keyInfo )

**Parameters** -> libRef CPM Library reference number (68k only).

-> encoding Constant that specifies the encoding type that was used to export the key. One of IMPORT\_EXPORT\_RAW, IMPORT\_EXPORT\_DER, or IMPORT\_EXPORT\_XML. See [Export Encoding Constants](#) for details about these formats.

-> importData The encoded data.

-> importDataLength The length of importData, in bytes.

<- hashInfo [APHashInfoStruct](#) that returns the imported hash info. The structure must be allocated before it's passed in.

**Result** The function returns errNone upon success. For other error codes, see [CPM Error Codes](#).

 **New**

## CPMLibImportKeyInfo

**Purpose** Imports a previously-exported key so that it (the key) can be used in subsequent operations.

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype** Err CPMLibImportKeyInfo ( UInt16 libRef,  
                  UInt8 encoding, UInt8 \*importData,  
                  UInt32 importDataLength, APKeyInfoType \*keyInfo )

**Parameters**

-> libRef	CPM Library reference number (68k only).
-> encoding	Constant that specifies the encoding type that was used to export the key. One of IMPORT_EXPORT_RAW, IMPORT_EXPORT_DER, or IMPORT_EXPORT_XML. See <a href="#">Export Encoding Constants</a> for details about these formats.

```
-> importData    The encoded data.  
-> importDataLength  
                  The length of importData, in bytes.  
<- keyInfo      APKeyInfoStruct that returns the imported  
                  key. The structure must be allocated before it's  
                  passed in.
```

**Result** The function returns `errNone` upon success. For other error codes, see [CPM Error Codes](#).



## CPMLibImportVerifyInfo

**Purpose** Imports a previously-exported verify info structure so that it can be used in subsequent operations.

**Declared In** `CPMLib68KInterface.h`, `CPMLibArmInterface.h`

**Prototype** `Err CPMLibImportVerifyInfo ( UInt16 libRef,  
 UInt8 encoding, UInt8 *importData,  
 UInt32 importDataLength,  
 APVerifyInfoType *keyInfo )`

**Parameters**

-> libRef	CPM Library reference number (68k only).
-> encoding	Constant that specifies the encoding type that was used to export the key. One of <code>IMPORT_EXPORT_RAW</code> , <code>IMPORT_EXPORT_DER</code> , or <code>IMPORT_EXPORT_XML</code> . See <a href="#">Export Encoding Constants</a> for details about these formats.
-> importData	The encoded data.
-> importDataLength	The length of importData, in bytes.

<- verifyInfo [APVerifyInfoStruct](#) that returns the imported verify info. The structure must be allocated before it's passed in.

**Result** The function returns `errNone` upon success. For other error codes, see [CPM Error Codes](#).



## CPMLibReleaseCipherInfo

**Purpose** Releases a [APCipherInfoStruct](#), allowing you to free it.

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype** Err CPMLibReleaseCipherInfo ( UInt16 libRef,  
APCipherInfoType \*cipherInfo )

**Parameters** -> libRef CPM Library reference number (68k only).  
-> cipherInfo A pointer to a [APCipherInfoStruct](#) that you want to release.

**Result** The function returns `errNone` upon success. For other error codes, see [CPM Error Codes](#).



## CPMLibReleaseHashInfo

**Purpose** Releases a [APKeyInfoStruct](#), allowing you to free it.

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype** Err CPMLibReleaseHashInfo ( UInt16 libRef,  
APHashInfoType \*hashInfo )

**Parameters** -> libRef CPM Library reference number (68k only).

-> hashInfo A pointer to a [APHashInfoStruct](#) that you want to release.

**Result** The function returns `errNone` upon success. For other error codes, see [CPM Error Codes](#).



## CPMLibReleaseKeyInfo

**Purpose** Releases a [APKeyInfoStruct](#), allowing you to free it.

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype** Err CPMLibReleaseKeyInfo ( UInt16 libRef,  
APKeyInfoType \*keyInfo )

**Parameters**

-> libRef	CPM Library reference number (68k only).
-> keyInfo	A pointer to the <a href="#">APKeyInfoStruct</a> that you want to release.

**Result** The function returns `errNone` upon success. For other error codes, see [CPM Error Codes](#).



## CPMLibReleaseVerifyInfo

**Purpose** Releases a [APVerifyInfoStruct](#), allowing you to free it.

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype** Err CPMLibReleaseVerifyInfo ( UInt16 libRef,  
APVerifyInfoType \*verifyInfo )

**Parameters**

-> libRef	CPM Library reference number (68k only).
-----------	--

-> verifyInfo A pointer to a [APVerifyInfoStruct](#) that you want to release.

**Result** The function returns `errNone` upon success. For other error codes, see [CPM Error Codes](#).



## CPMLibVerify

**Purpose** Verifies a message.

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype**

```
Err CPMLibVerify ( UInt16 libRef,
APKeyInfoType *keyInfo,
APVerifyInfoType *verifyInfo, UInt8 *inBuffer,
UInt32 inBufferLength, UInt8 *outBuffer,
UInt32 *outBufferLength, UInt8 *signature,
UInt32 signatureLength,
VerifyResultType *verifyResult )
```

**Parameters** -> `libRef` CPM Library reference number (68k only).

-> `keyInfo` An [APKeyInfoStruct](#) that represents the certificate's encryption key. Extracting the key data from the certificate and constructing the [APKeyInfoStruct](#) (through [CPMLibImportKeyInfo](#)) is the caller's responsibility.

## Cryptography Provider Manager

### CPM Functions

---

-> verifyInfo	A pointer to an <a href="#">APVerifyInfoStruct</a> that specifies the hash and cipher operations that should be performed during verification. This information is embedded as <a href="#">APHashInfoStruct</a> and <a href="#">APCipherInfoStruct</a> structures. If you want to use the default operations, allocate and zero the embedded structures. When the function returns, the structures will be populated with information describing the operations that were used.
-> inBuffer	A pointer to the message data.
-> inBufferLength	The length of inBuffer, in bytes.
<- outBuffer	A pointer to the buffer where the decrypted signature will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the decrypted data. If you don't care about the signature, pass NULL.
<-> outBufferLength	If outBuffer is NULL, set this to 0. Otherwise, you pass in the (allocated) length of outBuffer, in bytes. The function resets the argument to the amount of data that was actually copied into outBuffer. If the function returns cpmErrBufTooSmall, outBufferLength is set to the minimum buffer size that's needed to accommodate the verified data.
-> signature	A pointer to the message's signature. Extracting the signature from the message is the caller's responsibility
-> signatureLength	The length of the signature, in bytes.

<- verifyResult

An integer point that returns, by reference, the result of the verification. 0 means the message was verified; non-zero means it wasn't. The meaning of a non-zero return is defined by the algorithm provider.

**Result** The function returns `errNone` upon success. For other error codes, see [CPM Error Codes](#).

**Comments** This function performs a block verification operation. For stream verification, use [CPMLibVerifyInit](#).

Keep in mind that a direct return of `errNone` doesn't mean that the message has been verified. The verification status is returned by reference in `verifyResult`.



## CPMLibVerifyFinal

**Purpose** Finalizes a verify session.

**Declared In** CPMLib.h

**Prototype**

```
Err CPMLibVerifyFinal ( UInt16 libRef,
APKeyInfoType *keyInfo,
APVerifyInfoType *verifyInfo, UInt8 *inBuffer,
UInt32 inBufferLength, UInt8 *outBuffer,
UInt32 *outBufferLength, UInt8 *signature,
UInt32 signatureLength,
VerifyResultType *verifyResult )
```

**Parameters**

-> libRef	CPM Library reference number (68k only).
-> verifyInfo	A pointer to the <a href="#">APVerifyInfoStruct</a> that was returned by <a href="#">CPMLibVerifyInit</a> .
-> inBuffer	A pointer to the final buffer of message data, or NULL if there's no more data..

## Cryptography Provider Manager

### CPM Functions

---

-> inBufferLength  
The length of inBuffer, in bytes.

<- outBuffer A pointer to the buffer where the decrypted signature will be copied. The buffer must be allocated by the caller, and must be big enough to accommodate all of the decrypted data. If you don't care about the signature, pass NULL.

<-> outBufferLength  
If outBuffer is NULL, set this to 0. Otherwise, you pass in the (allocated) length of outBuffer, in bytes. The function resets the argument to the amount of data that was actually copied into outBuffer. If the function returns cpmErrBufTooSmall, outBufferLength is set to the minimum buffer size that's needed to accommodate the verified data.

-> signature  
A pointer to the message's signature. Extracting the signature from the message is the caller's responsibility

-> signatureLength  
The length of the signature, in bytes.

<- verifyResult  
An integer point that returns, by reference, the result of the verification. 0 means the message was verified; non-zero means it wasn't. The meaning of a non-zero return is defined by the algorithm provider.

**Result** The function returns errNone upon success. For other error codes, see [CPM Error Codes](#).

**Comments** This function feeds a final (optional) buffer of message into a verify stream that was previously initialized by [CPMLibVerifyInit](#) and augmented through successive calls to [CPMLibVerifyUpdate](#). The entire message is then verified, the results of the verification are returned in verifyResult, and the stream is closed.



## CPMLibVerifyInit

**Purpose** Initiates a streaming verify operation.

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype**

```
Err CPMLibVerify ( UInt16 libRef,  
APKeyInfoType *keyInfo,  
APVerifyInfoType *verifyInfo )
```

**Parameters**

-> libRef	CPM Library reference number (68k only).
-> keyInfo	An <a href="#">APKeyInfoStruct</a> that represents the certificate's encryption key. Extracting the key data from the certificate and constructing the <a href="#">APKeyInfoStruct</a> (through <a href="#">CPMLibImportKeyInfo</a> ) is the caller's responsibility.
-> verifyInfo	A pointer to an <a href="#">APVerifyInfoStruct</a> that specifies the hash and cipher operations that should be performed during verification. This information is embedded as <a href="#">APHASHINFOSTRUCT</a> and <a href="#">APCIPHERINFOSTRUCT</a> structures. If you want to use the default operations, allocate and zero the embedded structures. When the function returns, the structures will be populated with information describing the operations that were used.

**Result** The function returns `errNone` upon success. For other error codes, see [CPM Error Codes](#).

**Comments** This function initializes a streaming hash operation. To feed data to the stream, you call [CPMLibVerifyUpdate](#) followed by [CPMLibVerifyFinal](#). The "update" function is optional; the "final" function is mandatory. For a block hash, see [CPMLibVerify](#).



## New CPMLibVerifyUpdate

**Purpose** Sends message data to a streaming verify operation.

**Declared In** CPMLib68KInterface.h, CPMLibArmInterface.h

**Prototype** Err CPMLibVerifyUpdate ( UInt16 libRef,  
APKeyInfoType \*keyInfo,  
APVerifyInfoType \*verifyInfo )  
UInt8 \*inBuffer, UInt32 inBufferLength)

**Parameters**

-> libRef	CPM Library reference number (68k only).
-> keyInfo	The <a href="#">APKeyInfoStruct</a> that was used in the <a href="#">CPMLibVerifyInit</a> call.
-> verifyInfo	A pointer to the <a href="#">APVerifyInfoStruct</a> that was returned by <a href="#">CPMLibVerifyInit</a> .
-> inBuffer	A pointer to the message data.
-> inBufferLength	The length of inBuffer, in bytes.

**Result** The function returns errNone upon success. For other error codes, see [CPM Error Codes](#).

**Comments** This function feeds message data into the verify stream that was initialized by [CPMLibVerifyInit](#). When you're finished feeding data into the stream, call [CPMLibVerifyFinal](#) to close the stream and return the verification results.

## CPM Error Codes

The table below lists and explains the error codes that are returned by the CPM functions.

cpmErrAlreadyOpen	Returned by CPMLibOpen() to indicate that the CPM library has already been opened by your application. The open library remains open.
cpmErrNotOpen	All CPM functions (except CPMLibOpen()) expect the CPM library to be open. This code is returned if the library isn't open. To open the library, call CPMLibOpen().
cpmErrStillOpen	Returned by CPMLibClose() if the function was unable to close the library.
cpmErrNoProviders	Returned by CPMLibOpen() if the function was unable to locate (or load) any cryptography providers. Without a cryptography provider, none of the other CPM functions will work.
cpmErrNoBaseProvider	Returned by CPMLibOpen() if the function was unable to locate (or load) the default cryptography provider. (Note that this means that at least one non-default provider was found. If no providers were found, the function would have returned cpmErrNoProviders).
cpmErrProviderNotFound	Returned by CPMLibGenerateKey(), CPMLibEncrypt(), CPMLibDecrypt() and other algorithm-dependent functions if the requested provider couldn't be found.
cpmErrParamErr	Returned by a number of CPM functions when an argument is invalid (a pointer that points to an unallocated structure; a structure field that isn't properly set, and so on).
cpmErrOutOfMemory	Returned by CPMLibOpen() if the memory for a new library handle (and the structures it represents) couldn't be allocated

## Cryptography Provider Manager

### CPM Error Codes

---

cpmErrBufTooSmall	Returned by CPMLibExportKey() and CPMLibExportContext() functions if the storage buffer (allocated by the caller) isn't big enough to accommodate the key or context.
cpmErrBadData	Returned by algorithm functions (CPMLibEncrypt..., CPMLibDecrypt..., etc.) when the key or other required data is invalid.
cpmErrUnimplemented	Returned by functions that aren't currently implemented.
cpmErrUnsupported	Returned by functions that aren't currently supported.
cpmErrNoGlobals	Returned by functions that access global CPM data—functions such as CPMLibEnumerateProviders() and CPMLibGetProviderInfo()—when that data doesn't exist.
cpmErrKeyExists	Returned by CPMLibImportKey() when the key you're trying to import already exists.
cpmErrKeyNotFound	Returned by CPMLibExportKey() when the key you're trying to export doesn't exist.

---

# SSL Functions

---

This chapter describes the functions that are defined in the SSL library. These functions let you create and configure an SSL context, apply the SSL protocol to an existing socket, and send SSL data to and receive data over the socket.

But before you do any of that, you have to load and open the SSL library; something like this:

```
Err error;
UInt16 libRef;

if ( SyLibFind( kSslDBName, &libRef ) != 0 )
{
    error = SysLibLoad(kSslLibType, kSslLibCreator, &libRef);
}
/* error checking goes here. */

error = SslLibOpen( libRef );
...
```

---

After that, you typically...

1. ...create an [SslLib](#) through [SslLibCreate](#),
2. spawn an [SslContext](#) through [SslContextCreate](#),
3. set the context's socket (see the [SslContextSet\\_Socket](#) macro),
4. “open” the context through [SslOpen](#) (this enables the SSL protocol),
5. send and receive data through [SslRead](#) and [SslWrite](#),
6. and then close everything down ([SslClose](#), [SslContextDestroy](#), [SslLibDestroy](#), and [SslLibClose](#), in that order).

You can also use the more detailed [SslSend](#) and [SslReceive](#) functions to send and receive data. If you want to perform “streaming” reads—in which partial SSL records are read as the

## SSL Functions

### *SSL Attribute Functions and Macros*

---

data arrives—use [SslPeek](#) and [SslConsume](#). (For more on streaming, see the [ReadStreaming](#) attribute).

All functions described below are defined in `SslLib.h`.

## SSL Attribute Functions and Macros

An SSL context is defined, primarily, by the values of its SSL attributes. These attributes are set and retrieved through attribute-specific macros. For example, the [SslLibSet\\_InfoCallback](#) macro sets an [SslLib](#)'s [InfoCallback](#) attribute.

The attribute macros are defined in terms of a set of eight functions listed in this chapter. These functions—[SslLibSetLong](#), [SslContextSetLong](#), [SslLibSetPtr](#), [SslContextSetPtr](#), and so on—*can* be called directly, but it's suggested that you stick with the macros. If you want to call an attribute function directly, you have to identify the attribute by passing in one of the attribute constants.

The attribute macros and constants are described in [Chapter 82, “SSL Attributes and Macros,”](#) on page 2181.

## A Note on the Function Names

All of the SSL library functions have “Ssl” as a prefix. Furthermore, the expected Palm library functions ([SslLibOpen](#), [SslLibClose](#)) have an equally expected “SslLib” prefix. Unfortunately, another set of functions also uses “SslLib” as a prefix. These functions, [SslLibCreate](#), [SslLibDestroy](#), and so on, operate on instances of the [SslLib](#) data type, which type represents a generic SSL context.

Admittedly, “SslLib” is bad choice for the name of this data type. As pointed out in [Chapter 81, “SSL Structures and Data Types,”](#) when you see the [SslLib](#) data type, you should think “generic SSL context,” not “SSL library.”

# SSL Library Functions



## New SslClose

**Purpose** Shuts down an SSL session.

**Prototype** Err SslClose ( UInt16 libRef, SslContext \*context, UInt16 flags, UInt32 timeout )

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> context	The context that you want to close.
-> flags	Options that set the session's "shutdown" attributes.
-> timeout	Amount of time to wait for the final handshake messages from the server, in milliseconds.

**Result** errNone means success; for other codes, see [Chapter 83, “SSL Error Codes.”](#)

**Comments** By default, an SSL shutdown involves an exchange of messages with the server. As with [SslOpen](#), if this function times out, simply call it repeatedly until you get confirmation that the session has actually been closed (i.e. until the function returns errNone). Unlike with [SslClose](#), you needn't clear the flags when you re-call this function.

## SSL Functions

### SSL Library Functions

---

The flags set the SSL attributes that are used during the shutdown:

Flag	Meaning
sslCloseUseDefaultTimeout	If this flag is set, the current value of the <a href="#">IoTimeout</a> attribute is used as the function's timeout, overriding the timeout argument. If it isn't set, <a href="#">IoTimeout</a> is set to timeout and is used as the timeout for this function.
sslCloseDontSendShutdown	Sets the <a href="#">DontSendShutdown</a> attribute to 1. This suppresses the shutdown messages.
sslCloseDontWaitForShutdown	Sets the <a href="#">DontWaitForShutdown</a> attribute to 1. This tells the function to return after it sends a shutdown message to the server, thus ignoring the server's response.



## SslConsume

**Purpose** Removes data from a context's in-coming data buffer. Use after an [SslPeek](#) only.

**Prototype** `void SslConsume ( UInt16 libRef,  
SslContext *context, Int32 availableBytes )`

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> context	Context you want to look at.
-> availableBytes	The number of bytes to remove. This should always be the value that's returned to you by <a href="#">SslPeek</a> 's availableBytes argument.

**Comments** You call this function after calling [SslPeek](#) to remove the peeked at data.



## New **SslContextCreate**

**Purpose** Creates a new SSL context. The object is used to open, read, write, and close an SSL session.

**Prototype** Err SslContextCreate ( UInt16 libRef,  
SslLib \*contextTemplate, SslContext \*\*context )

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> contextTemplate	Pointer to an <a href="#">SslLib</a> object that will be used as the template for this context.
<- context	You pass in the address of an <a href="#">SslContext</a> pointer. The function allocates a new <a href="#">SslContext</a> object and points your pointer at it.

**Result** errNone means success; for other codes, see [Chapter 83, “SSL Error Codes.”](#)

**Comments** The attributes that have been set in contextTemplate are copied into context. These attributes don't include a network socket; setting the socket is typically the first thing you do with your context object. For instructions, see [Socket](#) in [Chapter 82, “SSL Attributes and Macros,”](#) on page 2181.

After you set its socket, the context object can be used to open an SSL session; see [SslOpen](#).

When you're finished using your context object, you close the session ([SslClose](#)) and then destroy the object by passing it to [SslContextDestroy](#).

## SSL Functions

### SSL Library Functions

---



New

## SslContextDestroy

**Purpose** Destroys an SSL context. You should close the object's SSL session (see [SslClose](#)) before destroying it.

**Prototype** Err SslContextDestroy ( UInt16 libRef,  
SslContext \*context)

**Parameters** -> libRef (68k only) SSL library reference number.  
-> context The context you wish to demolish.

**Result** errNone means success; for other codes, see [Chapter 83, "SSL Error Codes."](#)

**See Also** [SslContextCreate](#)



New

## SslContextGetLong

**Purpose** Returns the value of an integer-valued SSL attribute retrieved from a context.

**Prototype** Int32 SslLibGetLong ( UInt16 libRef,  
SslContext \*context, SslAttribute attribute )

**Parameters** -> libRef (68k only) SSL library reference number.  
-> context The context that contains the attribute.  
-> attribute Constant that represents the attribute that you want the value of. See "[SSL Attribute Constants](#)" in [Chapter 82, "SSL Attributes and Macros,"](#) on page 2181 for a list of attribute constants.

**Result** The attribute's value is returned directly.

**Comments** You should rarely need to invoke this function directly. Instead, use the attribute macros, as described in “[SSL Attribute Functions and Macros](#)” near the beginning of this chapter.

**See Also** [Chapter 82, “SSL Attributes and Macros,”](#) on page 2181,  
[SslContextSetLong](#), [SslContextSetPtr](#),  
[SslContextGetPtr](#)



## New

### SslContextGetPtr

**Purpose** Returns a pointer to an attribute that’s owned by a context template.

**Prototype**

```
Err SslLibGetPtr ( UInt16 libRef,
                   SslContext *context, SslAttribute attribute,
                   void **value )
```

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> context	The context that contains the attribute.
-> attribute	Constant that represents the attribute that you want the value of. “ <a href="#">SSL Attribute Constants</a> ” in <a href="#">Chapter 82, “SSL Attributes and Macros,”</a> on page 2181 for a list of attribute constants.
-> value	You pass in the address of a pointer; the function will point your pointer to the attribute’s data. The data’s type depends on the attribute. You mustn’t free or modify the pointed to data.

**Result** errNone means success; for other codes, see [Chapter 83, “SSL Error Codes.”](#)



## New **SslContextSetLong**

**Purpose** Modifies a context by changing the value of one of its integer-valued SSL attributes.

**Prototype**

```
Err SslLibSetLong ( UInt16 libRef,  
SslContext *context, SslAttribute attribute,  
Int32 value )
```

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> context	The context template you want to modify.
-> attribute	Constant that represents the attribute that you want to change. See “ <a href="#">SSL Attribute Constants</a> ” in <a href="#">Chapter 82, “SSL Attributes and Macros,”</a> on page 2181 for a list of attribute constants.
-> value	The attribute’s new (desired) value.

**Result** errNone means success; for other codes, see [Chapter 83, “SSL Error Codes.”](#)

**Comments** You should rarely need to invoke this function directly. Instead, use the attribute macros, as described in “[SSL Attribute Functions and Macros](#)” near the beginning of this chapter.

**See Also** [Chapter 82, “SSL Attributes and Macros,”](#) on page 2181, [SslContextSetPtr](#), [SslContextGetLong](#), [SslContextGetPtr](#)



## New **SslContextSetPtr**

**Purpose** Modifies a context template by changing the value of one of its pointer-valued SSL attributes.

**Prototype**

```
Err SslLibSetPtr ( UInt16 libRef,  
SslContext *context, SslAttribute attribute,  
void *value )
```

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> context	The context you want to modify.
-> attribute	Constant that represents the attribute that you want to change. See “ <a href="#">SSL Attribute Constants</a> ” in <a href="#">Chapter 82, “SSL Attributes and Macros,”</a> on page 2181 for a list of attribute constants.
-> value	A pointer to the attribute’s new (desired) value. The type of data that the pointer should point to is defined by the attribute.

**Result** errNone means success; for other codes, see [Chapter 83, “SSL Error Codes.”](#)

**Comments** You should rarely need to invoke this function directly. Instead, use the attribute macros, as described in “[SSL Attribute Functions and Macros](#)” near the beginning of this chapter.

**See Also** [Chapter 82, “SSL Attributes and Macros,”](#) on page 2181, [SslContextSetLong](#), [SslContextGetLong](#), [SslContextGetPtr](#)

**New** **SslFlush**

**Purpose** Flushes a context's out-going data buffer, sending the data to the network.

**Prototype** Err SslFlush ( Int16 libRef, SslContext \*context, Int32 \*outstanding )

**Parameters** -> libRef (68k only) SSL library reference number.

-> context Context you want to flush.

<- outstanding The number of bytes of data left in the buffer.

**Result** errNone means success; for other codes, see [Chapter 83, “SSL Error Codes.”](#)

**Comments** If the context is in autoflush mode (the [AutoFlush](#) attribute is 0), [SslWrite](#) and [SslSend](#) calls will write their data into the context's out-going data buffer, but the data won't actually be sent to the network. To “flush” this data, you have to explicitly call [SslFlush](#).

By not autoflushing, you can make multiple, small [SslWrite](#)/[SslSend](#) followed by an [SslFlush](#) and thereby improve network efficiency. Although the out-going buffer size is set to the value of the [WbufSize](#) attribute, the actual number of bytes that can be written by your application is somewhat smaller due to SSL overhead. After performing a write, you should check the value of the [WriteBufPending](#) attribute; if it's approaching the [WbufSize](#) value, you should call [SslFlush](#).

When the function returns, the outstanding value tells how many bytes of data are left in the buffer. If this value is non-zero, the next [SslWrite](#), [SslSend](#), or [SslFlush](#) call will attempt to write those bytes to the network.



## New **SslLibClose**

**Purpose** Closes a reference to the SSL library.

**Prototype** Err SslLibClose ( UInt16 refNum )

**Parameters** -> refNum      SSL library reference number (68k only).

**Result** errNone means success; for other codes, see [Chapter 83, “SSL Error Codes.”](#)

**Comments** You can only close a library that you've already opened. To open the SSL library, call [SslLibOpen](#).



## New **SslLibCreate**

**Purpose** Allocates and returns new [SslLib](#) object. The object is a template that describes a generic SSL context, and is used to create “real” context objects ([SslContext](#)).

**Declared In** SslLib.h

**Prototype** Err SslLibCreate ( UInt16 libRef,  
SslLib \*\*contextTemplate )

**Parameters** -> libRef      (68k only) SSL library reference number.

<- contextTemplate

You pass in the address of an [SslLib](#) pointer. The function allocates an [SslLib](#) object for you, and points your pointer at it.

**Result** errNone means success; for other codes, see [Chapter 83, “SSL Error Codes.”](#)

## SSL Functions

### SSL Library Functions

---

**Comments** You modify the context template (i.e the `SslLib` object) by calling the SSL attribute macros described in [Chapter 82, “SSL Attributes and Macros.”](#) You then use it as a template with which you create `SslContext` objects (see [SslContextCreate](#)). The latter objects are needed for actual SSL data transaction operations.

The `contextTemplate` that's returned by this function should ultimately be destroyed through [SslLibDestroy](#).

For more on `SslLib`, `SslContext`, and the relationship between them, see their descriptions in [Chapter 81, “SSL Structures and Data Types,”](#) on page 2163.



## SslLibDestroy

---

**Purpose** Destroys an `SslLib` object and frees all the memory it allocated.

**Prototype**

```
void SslLibDestroy ( UInt16 libRef,  
                     SslLib *contextTemplate )
```

**Parameters**

- > `libRef` (68k only) SSL library reference number.
- > `contextTemplate` The `SslLib` you want to destroy, as previously returned by [SslLibCreate](#).

**Result** `errNone` means success; for other codes, see [Chapter 83, “SSL Error Codes.”](#)

**Comments** The `SslContext` objects that were spawned by this `SslLib` maintain a reference to their spawner. Because of this, you shouldn't destroy an `SslLib` until you've destroyed all of its contexts.



## New **SslLibOpen**

**Purpose** Opens a reference to the SSL library.

**Prototype** Err SslLibOpen ( UInt16 libRef )

**Parameters** -> libRef (68k only) SSL library reference number.

**Result** errNone means success; for other codes, see [Chapter 83, “SSL Error Codes.”](#)

**Comments** When you’re done using the SSL library, you must close it through [SslLibClose](#)



## New **SslLibGetLong**

**Purpose** Returns the value of an integer-valued SSL attribute retrieved from a context template.

**Prototype** Int32 SslLibGetLong ( UInt16 libRef,  
SslLib \*contextTemplate, SslAttribute attribute )

**Parameters** -> libRef (68k only) SSL library reference number.

-> contextTemplate

The context template that contains the attribute.

-> attribute

Constant that represents the attribute that you want the value of. See “[SSL Attribute Constants](#)” in [Chapter 82, “SSL Attributes and Macros,”](#) on page 2181 for a list of attribute constants.

**Result** The attribute’s value is returned directly.

## SSL Functions

### SSL Library Functions

---

**Comments** You should rarely need to invoke this function directly. Instead, use the attribute macros, as described in “[SSL Attribute Functions and Macros](#)” near the beginning of this chapter.

**See Also** [Chapter 82, “SSL Attributes and Macros,”](#) on page 2181,  
[SslLibSetLong](#), [SslLibSetPtr](#), [SslLibGetPtr](#)



## SslLibGetPtr

**Purpose** Returns a pointer to an attribute that’s owned by a context template.

**Prototype** Err SslLibGetPtr ( UInt16 libRef,  
SslLib \*contextTemplate, SslAttribute attribute,  
void \*\*value )

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> contextTemplate	The context template that contains the attribute.
-> attribute	Constant that represents the attribute that you want the value of. “ <a href="#">SSL Attribute Constants</a> ” in <a href="#">Chapter 82, “SSL Attributes and Macros,”</a> on page 2181 for a list of attribute constants.
-> value	You pass in the address of a pointer; the function will point your pointer to the attribute’s data. The data’s type depends on the attribute. You mustn’t free or modify the pointed to data.

**Result** errNone means success; for other codes, see [Chapter 83, “SSL Error Codes.”](#)

**Comments** You should rarely need to invoke this function directly. Instead, use the attribute macros, as described in “[SSL Attribute Functions and Macros](#)” near the beginning of this chapter.

**See Also** [Chapter 82, “SSL Attributes and Macros,” on page 2181](#), [SslLibSetLong](#), [SslLibGetLong](#), [SslLibSetPtr](#)



## SslLibSetLong

**Purpose** Modifies a context template by changing the value of one of its integer-valued SSL attributes.

**Prototype** Err SslLibSetLong ( UInt16 libRef,  
SslLib \*contextTemplate, SslAttribute attribute,  
Int32 value )

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> contextTemplate	The context template you want to modify.
-> attribute	Constant that represents the attribute that you want to change. See “ <a href="#">SSL Attribute Constants</a> ” in <a href="#">Chapter 82, “SSL Attributes and Macros,” on page 2181</a> for a list of attribute constants.
-> value	The attribute’s new (desired) value.

**Result** errNone means success; for other codes, see [Chapter 83, “SSL Error Codes.”](#)

**Comments** You should rarely need to invoke this function directly. Instead, use the attribute macros, as described in “[SSL Attribute Functions and Macros](#)” near the beginning of this chapter.

**See Also** [Chapter 82, “SSL Attributes and Macros,” on page 2181](#), [SslLibSetPtr](#), [SslLibGetLong](#), [SslLibGetPtr](#)

**New**

## SslLibSetPtr

**Purpose** Modifies a context template by changing the value of one of its pointer-valued SSL attributes.

**Prototype**

```
Err SslLibSetPtr ( UInt16 libRef,  
SslLib *contextTemplate, SslAttribute attribute,  
void *value )
```

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> contextTemplate	The context template you want to modify.
-> attribute	Constant that represents the attribute that you want to change. See “ <a href="#">SSL Attribute Constants</a> ” in <a href="#">Chapter 82, “SSL Attributes and Macros,”</a> on page 2181 for a list of attribute constants.
-> value	A pointer to the attribute’s new (desired) value. The type of data that the pointer should point to is defined by the attribute.

**Result** errNone means success; for other codes, see [Chapter 83, “SSL Error Codes.”](#)

**Comments** You should rarely need to invoke this function directly. Instead, use the attribute macros, as described in “[SSL Attribute Functions and Macros](#)” near the beginning of this chapter.

**See Also** [Chapter 82, “SSL Attributes and Macros,”](#) on page 2181, [SslLibSetLong](#), [SslLibGetLong](#), [SslLibGetPtr](#)

**Comments** You should rarely need to invoke this function directly. Instead, use the attribute macros, as described in “[SSL Attribute Functions and Macros](#)” near the beginning of this chapter.

**See Also** [Chapter 82, “SSL Attributes and Macros,”](#) on page 2181,  
[SslContextGetLong](#), [SslContextSetLong](#),  
[SslContextSetPtr](#)



## SslOpen

**Purpose** Opens an SSL session, possibly as a continuation of a previous session.

**Prototype** Err SslOpen ( UInt16 libRef, SslContext \*context, UInt16 flags, UInt32 timeout )

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> context	The context object that will configure and own the session.
-> flags	Options that set various SSL attributes. The most important flags are <code>sslOpenModeSsl</code> and <code>sslOpenModeClear</code> . These mutually exclusive flags tell the session to use SSL or cleartext, respectively.
-> timeout	Amount of time to wait for the handshake confirmation before giving up, in milliseconds.

**Result** `errNone` means success; for other codes, see [Chapter 83, “SSL Error Codes.”](#)

**Comments** Calling `SslOpen` is essentially the same as setting the attributes that correspond to the function’s flags:

## SSL Functions

### SSL Library Functions

---

Flag	Meaning
sslOpenModeClear	Sets the context's <a href="#">Mode</a> attribute to <code>sslModeClear</code> (cleartext).
sslOpenModeSsl	Sets the context's <a href="#">Mode</a> attribute to <code>sslModeSslClient</code> (SSL is enabled). This overrides <code>sslOpenModeClear</code> .
sslOpenNoAutoFlush	Sets the <a href="#">AutoFlush</a> attribute to 0. This suppresses auto-flushing so that <a href="#">SslWrite</a> / <a href="#">SslSend</a> data is cached until you call <a href="#">SslFlush</a> .
sslOpenNewConnection	Sets the context's <a href="#">SslSession</a> attribute to NULL. This will force a full SSL handshake. If you don't set this flag, the previous <a href="#">SslSession</a> will be used (if possible), and the handshake is truncated.
sslOpenBufferedReuse	Sets the <a href="#">BufferedReuse</a> attribute to 1. This provides an even greater savings during a (truncated) handshake.
sslOpenUseDefaultTimeout	If this flag is set, the current value of the <a href="#">IoTimeout</a> attribute is used as the function's timeout, overriding the <code>timeout</code> argument. If it isn't set, <a href="#">IoTimeout</a> is set to <code>timeout</code> and is used as the timeout for this function.
sslOpenDelayHandshake	Instead of sending the final handshake message, cache it until the write buffer is flushed (through an autoflush, or through an <a href="#">SslFlush</a> call). There is no attribute that corresponds to this flag.

If the function times out, you should simply call it again—timing out isn't a fatal error. Each time you call `SslOpen` (after a timeout), the handshake continues from where it timed out. However, it's important that when you re-call `SslOpen`, that you clear the `flags` argument. The attributes that they affect will already have been set; setting them a second time could cause the context to be reset, which will cause the handshake to start from the beginning.



## New **SslPeek**

**Purpose** Lets you look at in-coming data without actually “consuming” it.

**Prototype**

```
Err SslPeek ( UInt16 libRef, SslContext *context,
               void **data, Int32 *availableBytes,
               Int32 maxAvailable)
```

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> context	Context you want to look at.
<- data	Pass in the address of a pointer; the function sets your pointer to point to the context's in-coming data buffer.
<- availableBytes	The function returns the smaller of <b>a</b> ) the number of bytes of data that are waiting to be read or <b>b</b> ) the value of maxAvailable.
-> maxAvailable	Sets a limit on the availableBytes value.

**Result** Always returns 0.

**Comments** This function is similar to [SslRead](#), but rather than copy data into your buffer, it gives you direct access to the context's in-coming data buffer. If the buffer is empty, data is read from the network until there are bytes available. After each successful `SslPeek` call, you should call [SslConsume](#) to clear the “peeked at” data. If you don't consume after a peek, the next `SslPeek` will return a pointer to the same data.

`SslPeek` is meant to be used in contexts that have the [ReadStreaming](#) attribute set.

## SSL Functions

### *SSL Library Functions*

---



#### New SslRead

**Purpose** Copies a buffer of in-coming data.

**Prototype**

```
Int16 SslRead ( UInt16 libRef,
                 SslContext *context, void *buffer,
                 UInt16 bufferLength, Err *error )
```

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> context	Context that owns the session.
<- buffer	This is where the retrieved data will be copied. The buffer must be allocated by the caller.
-> bufferLength	The length of buffer, in bytes.
<- error	Error code as listed in see <a href="#">Chapter 83, “SSL Error Codes.”</a>

**Result** Returns the number of bytes that were actually read. A return of 0 means the socket was shut down by the server. If the return is -1, look in `error` for the precise error code.

**Comments** This is a convenient cover for [SslReceive](#). It ignores the address of the sender, doesn't set any socket flags, and uses the context's current timeout setting (the [IoTimeout](#) attribute).



## New **SslReceive**

**Purpose** Copies a buffer of in-coming data.

**Prototype**

```
Int16 SslReceive ( UInt16 libRef,
SslContext *context, void *buffer,
UInt16 bufferLength, UInt16 flags,
void *fromAddress, UInt16 *fromAddressLength,
Int32 timeout, Err *error )
```

<b>Parameters</b>	-> libRef	(68k only) SSL library reference number.
	-> context	Context that owns the session.
	<- buffer	This is where the retrieved data will be placed.
	-> bufferLength	The length of buffer, in bytes.
	-> flags	Options that are applied to (and stored by) the socket. See <a href="#">I/O Flags</a> in <a href="#">Chapter 61, “Net Library”</a> on page 1413 for a description of these options.
	<- fromAddress	Optionally returns the address of the sender. The fromAddress buffer must be allocated by the caller. If you don't want to retrieve the address, pass a NULL pointer.
	<-> fromAddressLength	On input, you set *fromAddressLength to the size of the fromAddress buffer, in bytes. Upon return, *fromAddressLength is set to the actual size of the buffer. If you don't want to retrieve the address, pass a NULL pointer.
	-> timeout	Maximum timeout in system ticks; -1 means wait forever.

## SSL Functions

### SSL Library Functions

---

<- error              Error code as listed in see [Chapter 83, “SSL Error Codes.”](#)

**Result**    Returns the number of bytes that were actually read. A return of 0 means the socket was shut down by the server. If the return is -1, look in `error` for the precise error code.

**Comments**    Unlike in `SslSend`, the address that you supply (`fromAddress`) doesn't overwrite the socket's address. This is a cover for the Network library's [`NetLibDmReceive`](#) function.

For a shorthand version of this function, see [`SslRead`](#).



## SslSend

---

**Purpose**    Writes a buffer of data to the network, or buffers it for a later send.

**Prototype**

```
Int16 SslSend ( UInt16 libRef,
                 SslContext *context, void *buffer,
                 UInt16 bufferLength, UInt16 flags,
                 void *toAddress, UInt16 toAddressLength,
                 Int32 timeout, Err *error )
```

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> context	Context that owns the SSL session.
-> buffer	A pointer to the data that you want to write.
-> bufferLength	The amount of data you want to write, in bytes.
-> flags	Options that are applied to (and stored by) the socket. See <a href="#"><u>I/O Flags</u></a> in <a href="#"><u>Chapter 61, “Net Library</u></a> , on page 1413 for a description of these options.
-> toAddress	Address of the recipient. If this is a <a href="#"><u>NetSocketAddrType</u></a> , the socket's address is set to this value. If it's 0, the socket's current address is used.

-> `toAddressLength` Size of `*toAddress`, in bytes.  
-> `timeout` Maximum timeout in system ticks; -1 means wait forever.  
<- `error` Error code as listed in see [Chapter 83, “SSL Error Codes.”](#)

**Result** Returns the number of bytes that were actually sent. A return of 0 means the socket was shut down by the server. If the return is -1, look in `error` for the precise error code.

**Comments** If the context has the [AutoFlush](#) attribute enabled, the data is immediately written to the network; otherwise, it's cached until [SslFlush](#) is called. This function is a cover for the Network library's [NetLibSend](#) function.

For a shorthand version of this function, see [SslWrite](#).

  
**New**

## SslWrite

---

**Purpose** Writes a buffer of data to the network, or caches it in anticipation of a flush.

**Prototype** `Int16 SslWrite ( UInt16 libRef,  
SslContext *context, void *buffer,  
UInt16 bufferLength, Err *error )`

**Parameters**

-> <code>libRef</code>	(68k only) SSL library reference number.
-> <code>context</code>	Context that owns the session.
-> <code>buffer</code>	A pointer to the data that you want to write.
-> <code>bufferLength</code>	The amount of data you want to write, in bytes.

## SSL Functions

### *Application-Defined Functions*

---

<- error                  Error code as listed in see [Chapter 83, “SSL Error Codes.”](#)

**Result**    Returns the number of bytes that were actually sent. A return of 0 means the socket was shut down by the server. If the return is -1, look in `error` for the precise error code.

**Comments**    This is a convenient cover for [`SslSend`](#). It doesn’t set any socket flags, uses the socket’s current address, and uses the context’s current timeout setting (the [`IoTimeout`](#) attribute).

## Application-Defined Functions



### New **SslCallbackFunc**

---

**Purpose**    Prototype for SSL information and verification callback functions.

**Prototype**    `typedef Int32 (*SslCallbackFunc) (SslCallback *callbackStruct, Int32 command, Int32 flavor, void *info);`

**Parameters**

-> <code>callbackStruct</code>	A structure that contains information about this callback function, including application-defined data that can be used in the function’s implementation. See <a href="#"><code>SslCallback</code></a> in <a href="#">Chapter 81, “SSL Structures and Data Types,”</a> on page 2163.
-> <code>command</code>	A constant that represents the general reason for the function’s invocation.
-> <code>flavor</code>	A constant that refines the command value.

-> `info` Additional data that may be needed by the callback. The type of data that's passed depends on the command/flavor values.

**Result** The function should return `errNone` (0) upon success, otherwise, it should return one of the error codes described in [Chapter 83, “SSL Error Codes.”](#) A non-zero return value here is returned to the SSL function that caused the callback to be invoked.

**Comments** For every context, you can install two callback functions: An “info” callback and a “verify” callback:

- The info callback is typically used for debugging, displaying data, tracking progress, and so on. It's invoked at well-defined junctures as the context is working.
- The verify callback is called when a certificate is being verified. It's expected to handle error situations.

To install a callback, you create an [`SslCallback`](#) structure, populate the necessary fields, and then use it to set the value of the [`InfoCallback`](#) or [`VerifyCallback`](#) attribute. When you set the info callback, you also have to set the [`InfoInterest`](#) attribute to register for specific events (which will show up in the flavor argument when your callback is invoked). The verify callback doesn't require similar event registration.

Setting the info callback in an [`SslContext`](#) looks a little bit like this:

---

```
/* Create an SslCallback structure, and point its
 * 'callback' field to your function ('MyInfoCallbackFunc').
 */
SslCallback infoCallback;
infoCallback.callback = MyInfoCallbackFunc;

/* Set the InfoCallback attribute. */
SslContextSet_InfoCallback( sslContext, &infoCallback );

/* Set the events you're interested in hearing about. */
SslContextSet_InfoInterest( sslContext,
                           sslFlgInfoAlert | sslFlgInfoHandshake | sslFlgInfoIo);
```

---

When you set the [`InfoCallback`](#) or [`VerifyCallback`](#) attribute, the [`SslCallback`](#) structure is copied into the object. Furthermore,

## SSL Functions

### *Application-Defined Functions*

---

when you spawn an [SslContext](#) from an [SslLib](#), the callback structures are copied into the new object.

### Commands and Flavors

Taken together, the `command` and `flavor` arguments tell a callback why it's being called. There are three command groups:

- The common command values (`sslCmdNew`, `sslResetNew`, and `sslCmdFree`) apply to both info and verify callbacks. These commands don't use flavor arguments.
- `sslCmdInfo` is used for info callbacks only. The `flavor` argument indicates the reason for the callback.
- Similarly, `sslCmdVerify` is used for verify callbacks only (and flavor refines the command's meaning).

### *Common Commands*

The common commands are:

Command	Meaning
<code>sslCmdNew</code>	The callback function was just copied into an object. (More accurately, the <a href="#">SslCallback</a> that contains the function was copied.) This happens when you set the <a href="#">InfoCallback</a> or <a href="#">VerifyCallback</a> attribute, or when you spawn an <a href="#">SslContext</a> .
<code>sslCmdReset</code>	The context that contains this callback was just reset ( <a href="#">SslContext</a> only; see the <a href="#">Mode</a> attribute for a brief explanation of the SSL reset).
<code>sslCmdFree</code>	The <a href="#">SslCallback</a> structure is about to be freed. This usually means that the <a href="#">SslLib</a> or <a href="#">SslContext</a> that owns the structure is being destroyed.

When a callback function is invoked with one of these common commands, the `flavor` argument is 0 (i.e. undefined) and `info` is `NULL`.

### ***sslCmdInfo Flavors***

As mentioned above, the `sslCmdInfo` command argument is only sent to the info callback (keep in mind, however, that the info callback may also receive any of the common commands).

The flavor values that are associated with the `sslCmdInfo` command are:

<b>Flavor</b>	<b>Meaning</b>
<code>sslArgInfoHandshake</code>	Notification of an SSL state change. Read the <a href="#">HsState</a> attribute to determine the new state.  The callback's <code>info</code> parameter is set to NULL.
<code>sslArgInfoAlert</code>	Notification of an SSL alert. Your callback can read the <a href="#">LastAlert</a> attribute to determine which alert was received.  The callback's <code>info</code> parameter is set to NULL.
<code>sslArgInfoReadBefore</code> <code>sslArgInfoReadAfter</code> <code>sslArgInfoWriteBefore</code> <code>sslArgInfoWriteAfter</code>	Notifications that are sent just before and just after data is read from or written to the network.  The <code>info</code> argument is an <a href="#">SslIoBuf</a> structure that contains the data.
<code>sslArgCert</code>	Notification that the server's certificate chain has been verified.  The <code>info</code> argument is an <a href="#">SslExtendedItems</a> structure that contains the server's certificate.

### ***sslCmdVerify Flavors***

The verify callback function is called with the `sslCmdVerify` command when an error occurs during certificate verification, and

## SSL Functions

### *Application-Defined Functions*

---

when the certificate is successfully verified. The associated flavor values are:

Flavor	Meaning
sslErrVerifyBadSignature	The certificate's signature is invalid
sslErrVerifyNoTrustedRoot	A trusted certificate store (necessary for certificate verification) couldn't be found
sslErrVerifyNotAfter	The certificate has expired.
sslErrVerifyNotBefore	The certificate is too early (the timestamp window is in the future).
sslErrVerifyConstraintViolation	The certificate violates an X509 extension
sslErrVerifyUnknownCriticalExtension	An X509 extension (that's marked as "critical") isn't understood by the certificate verification routines.y
sslErrOk	The certificate was successfully verified.

Note that these constants are also used as error codes. If your verify function can't fix the problem, it should return the command argument as an error.

In all cases, the info argument is an [SslVerify](#) structure that contains information about the certificate that's being verified.

# SSL Structures and Data Types

---

This chapter describes the structures and data types that are used by SSL.

All elements described below are defined in `SslLib.h`.

## SSL Data Types



### SslAttribute

---

The `SslAttribute` data type is used to cast the SSL attribute constants that are passed to the attribute-setting functions.

```
typedef UInt32 SslAttribute;
```

In general, you should use the attribute-setting macros rather than the functions. The attribute upon which a macro operates is embedded in its name, so you rarely have to deal with `SslAttribute` types.

For more information on the relationship between the attribute-setting functions and macros, see [Chapter 80, “SSL Functions,”](#) on page 2135. For a list of the macros themselves, go to [Chapter 82, “SSL Attributes and Macros,”](#) on page 2181.



### SslContext

---

`SslContext` is the data type for a private structure that holds all the information, or *context*, associated with the SSL protocol that will be used in an SSL session. It contains flags that govern how the

## SSL Structures and Data Types

### SSL Data Types

---

SSL protocol will operate, read and write buffers where SSL packets are assembled and disassembled, various structures that are created as part of the SSL handshake, and so on.

The data type is declared in `SslLib.h` as:

```
typedef struct SslContext_st SslContext;
```

The lifetime of an `SslContext` follows this pattern:

1. **Creation.** You never allocate your own `SslContext` objects; instead, you first create an `SslLib` (which see) and pass it to the `SslContextCreate` function. The `SslLib` acts as a template that's used by the function to create and configure a new `SslContext`. This "configuration" consists of copying the `SslLib`'s **SSL attributes** into the new `SslContext`.
2. **Configuration.** After you get your hands on the new `SslContext`, you can refine its SSL attributes by calling the SSL attribute macros.
3. **Socket specification.** The one essential attribute that you must set is the context's socket. The template doesn't contain a reference to a socket, so if you want to actually use the `SslContext` object's that you create, you have to explicitly set its socket. See `Socket` in Chapter 82, "SSL Attributes and Macros."
4. **SSL sessions.** You then pass the `SslContext` to `SslOpen`, which creates a new SSL session. The `SslContext` is used as a cookie in session operations such as `SslRead` and `SslClose`. When you're done with the session, you call `SslClose`. The same `SslContext` can be reused in successive sessions, but only one at a time.
5. **Destruction.** When you're done using the SSL context, you hand the object back to the system through `SslContextDestroy`.



### SslLib

---

`SslLib` is the data type for a private structure that describes a generic SSL context. It's declared in `SslLib.h` as:

```
typedef struct SslLib_st SslLib;
```

---

**NOTE:** `SslLib` is a misleading name for this data type. The type *doesn't* represent the SSL library, nor does it serve any explicit library-related function. Where you see “`SslLib`” you should think “generic SSL context.”

---

After you create and modify an `SslLib` object, you use it as a template to create “real” SSL context objects (type [SslContext](#)). When you’re done with the `SslLib`, you ask the SSL library to destroy it.

Functionally, the lifetime of an `SslLib` follows this pattern:

1. **Creation.** You create the object through [SslLibCreate](#).
2. **Modification.** Since the structure that the `SslLib` represents is private, you can’t touch its fields directly. Instead, you use a set of macros (defined in `SslLibMac.h`) to fine-tune the object. These macros set the values of the **SSL attributes** that the `SslLib` contains. The SSL attributes and the macros that you use to set and get their values are described in [Chapter 82, “SSL Attributes and Macros,”](#) on page 2181.
3. **Context creation.** When you’re satisfied with your `SslLib`’s attribute configuration, you use it to create new [SslContext](#) objects through the [SslContextCreate](#) function. The `SslLib`’s attributes are copied into the new [SslContext](#). Subsequent changes to the `SslLib` won’t affect the [SslContext](#) objects that you’ve already created.
4. **Destruction.** When you’re done with your `SslLib` and all of its [SslContexts](#) you should destroy it by passing it to [SslLibDestroy](#). Note that each [SslContext](#) object keeps a reference to the `SslLib` that calved it, so you mustn’t destroy an `SslLib` until after its [SslContext](#) objects have been freed.

You can create as many `SslLib` objects as you want, but you typically only create one per application.

## SSL Structures and Data Types

### SSL Structures

---

## SSL Structures



### SslCallback

The SslCallback structure contains information about a callback function that's invoked during SSL operations.

```
typedef struct {
    void *reserved;
    SslLibCallbackFunc callback;
    void *data;
    SslContext *ssl;
} SslCallback
```

The fields are:

callback	A pointer to a callback function.
data	A pointer to data that's passed to the callback function when it's invoked.
ssl	The SSL context to which the callback applies, or NULL if it's not specific to a context. You never set this field; it's set for you when the callback is copied into an <a href="#">SslContext</a> object.
reserved	Reserved for future use.

There are two types of callback functions: info and verify. The info callback is called as data is being read, when the server sends an alert, and so on. The verify callback is called when a certificate is being verified.

To register a callback function, you create an `SslLibCallback` structure, fill out the `callback` and `data` fields, and then use the structure to set the value of the [InfoCallback](#) or

[VerifyCallback](#) attribute. To set the attribute you call one of these macros:

- [SslLibSet\\_InfoCallback](#) sets the [InfoCallback](#) attribute of an [SslLib](#) object.
- [SslLibSet\\_VerifyCallback](#) sets the [VerifyCallback](#) attribute of an [SslLib](#) object.
- [SslContextSet\\_InfoCallback](#) sets the [InfoCallback](#) attribute of an [SslContext](#) object.
- [SslContextSet\\_VerifyCallback](#) sets the [VerifyCallback](#) attribute of an [SslContext](#) object.

When you register an info callback you must also set the [InfoInterest](#) attribute by calling [SslLibSet\\_InfoInterest](#) or [SslContextSet\\_InfoInterest](#) macro. The [InfoInterest](#) attribute contains a list of the events that your info callback is interested in.

As with (nearly) all SSL attributes, the callbacks that you set in an [SslLib](#) are copied into the [SslContext](#) objects that it spawns.

You can use the same [SslLibCallback](#) structure to set more than one callback function. After you've called a callback-registering macro, you can free the original [SslLibCallback](#) structure.

The protocol for the callback functions is described in [SslCallbackFunc](#), in [Chapter 80, “SSL Functions.”](#)



---

## SslCipherSuiteInfo

Structure that contains information about the cipher suite that's being used. The structure is stored in the [CipherSuiteInfo](#) attribute.

```
ttypedef struct SslCipherSuiteInfo_st {  
    UInt8  cipherSuite[2];  
    UInt16 cipher;  
    UInt16 digest;  
    UInt16 keyExchange;  
    UInt16 authentication;  
    UInt16 version;
```

## SSL Structures and Data Types

### SSL Structures

---

```
    UInt16 cipherBitLength;
    UInt16 cipherKeyLength;
    UInt16 keyExchangeLength;
    UInt16 authenticationLength;
    UInt16 export;
} SslCipherSuiteInfo;
```

The fields are:

cipherSuite	A two-byte value that represents the current cipher suite. The possible values, encoded as constants, are: 0: No cipher suite is being used. <code>sslCs_RSA_RC4_56_SHA1</code> : Secure Hash Algorithm-1, 56-bit <code>sslCs_RSA_RC4_128_SHA1</code> : Secure Hash Algorithm-1, 128-bit. <code>sslCs_RSA_RC4_40_MD5</code> : Rivest Message Digest 5, 40-bit. <code>sslCs_RSA_RC4_128_MD5</code> : Rivest Message Digest 5, 128-bit.
cipher	A constant that represents the cipher that's being used for this connection: <code>sslCsiCipherNull</code> : No cipher currently set. <code>sslCsiCipherRc4</code> : RSA RC4.
digest	A constant that represents the message digest format.: <code>sslCsiDigestNull</code> : No message digest. <code>sslCsiDigestMd2</code> : Rivest Message Digest 2. <code>sslCsiDigestMd5</code> : Rivest Message Digest 5. <code>sslCsiDigestSha1</code> : Secure Hash Algorithm-1.
keyExchange	A constant that represents the key exchange type that's being used: <code>sslCsiKeyExchNull</code> : No key exchange. <code>sslCsiKeyExchRsa</code> : RSA.

authentication	A constant that represents the authentication type that's being used:  sslCsiAuthNull: No key exchange. sslCsiAuthRsa: RSA.
version	The SSL version number.
cipherBitLength	The length of the material used for encryption key generation, in bits. For export ciphers this will be either 40 or 56 bits.
cipherKeyLength	The length of the encryption key that's generated, in bits. For an export RC4 cipher, the cipherKeyLength is 128.
keyExchangeLength	The length of the public key used to establish a shared secret, in bits.
authenticationLength	The length of the public key used to ensure that the key exchange wasn't tampered with, in bits. For export ciphers, the keyExchangeLength is often shorter than the authenticationLength.
export	A boolean value: 1 if an export cipher is being used; 0 otherwise.

The list of possible cipher suites can be retrieved from the [CipherSuites](#) attribute. The suite that's currently being used is given by the [CipherSuite](#) attribute.

 **New**

## SslExtendedItem

Structure that's used to describe a single certificate-related item. Every SslExtendedItem has a distinct (pre-defined) "extended item type." The location of the item's data is indicated by the structure—the data isn't stored in the structure.

You never create SslExtendedItem structures yourself. The objects that relate to a particular certificate are collected into an

## SSL Structures and Data Types

### SSL Structures

---

[SslExtendedItems](#) structure and returned to you (principally) by your verify callback function.

```
typedef struct SslExtendedItem_st {
    UInt16 type;
    UInt16 field;
    UInt16 dataType;
    UInt16 len;
    UInt32 offset;
} SslExtendedItem;
```

The fields are:

---

type	The extended item type, one of:  sslExItemTypeX509 : X.509 Certificate sslExItemTypeRSA : RSA public key sslExItemTypeRDN : An X.509 Relative Distinguished Name (RDN). This is the certificate's name. Each certificate contains two names, the <i>Subject</i> of the certificate and the <i>Issuer</i> of the certificate. Both are encoded as RDNs that contain multiple fields.  sslExItemTypeX509Ex : X.509 certificates can contain multiple "extensions." This type is used to specify that the item is a certificate extension.
field	Type-specific value.
dataType	The encoding for the item's data. For the SSL library, the value is one of the ASN.1 encoding types, as listed in <code>SslLibAsn1.h</code> . The encoding is relevant if you're trying to display the data bytes.

---

len	Number of data bytes.
offset	Location of the data itself, as an offset in bytes from the beginning of the <a href="#">SslExtendedItems</a> structure that contains this SslExtendedItem.

---



## SslExtendedItems

Structure that contains a set of related data items that pertain to an SSL certificate or other cryptographic entity. The [SslExtendedItems](#) structure is used as part of the [SslVerify](#) structure, and is used as the type of the [PeerCert](#) attribute.

The [SslExtendedItems](#) structure is used by the SSL library to return information to your application; you never create and populate an [SslExtendedItems](#) structure yourself.

```
typedef struct SslExtendedItems_st
{
    UInt32 length;
    UInt32 num;
    SslExtendedItem eitem[1];
} SslExtendedItems;
```

The fields are:

---

length	Total length of the structure, in bytes.
num	Number of elements in the eitem array.
eitem	Individual data items (these are the “related data items” mentioned above). The array contains num elements.

---

The items in the eitem array needn’t all be the same “extended item type” (as defined in [SslExtendedItem](#)). For example, the [SslExtendedItems](#) structure for a certificate typically contains an [sslExItemTypeX509](#) item, an [sslExItemTypeRSA](#), and an [sslExItemType](#) (for the subject name).

## SSL Structures and Data Types

### SSL Structures

---



## New SslIoBuf

Structure that represents a data I/O operation. It's passed to your info callback function as data is being read or written.

```
typedef struct {
    SslContext *ssl;
    UInt8 *ptr;
    UInt32 outNum;
    UInt32 inNum;
    UInt32 max;
    UInt32 err;
    UInt32 flags;
} SslIoBuf;
```

You can ask for the info callback function to be invoked just before data is read, just after it's read, just before it's written, and just after it's written. You set the [InfoInterest](#) attribute to register for these four events.

Most of the SslIoBuf fields depend on which of these four events is being described:

### Fields

---

ctx	The SSL context that performed the operation.
ptr	A pointer to the data that was just read, or that's about to be written. For "before read" and "after write" events, the buffer should be empty.
outNum	The number of bytes that were just read or written. For the "before" events, this field is set to 0.
inNum	The number of bytes that are about to be read or written. For the "before read" case, this is the minimum number of bytes that the context wants to read—the actual number of bytes read may be larger.

---

max	This is valid for the “before read” event only. It’s set to the maximum number of bytes that can be read during the current operation.
error	This is set to the error code that was returned by a read operation. It’s only valid for the “after” events.
flags	Currently ignored.

---

The information in this structure is provided for debugging and informational purposes only. You could use it, for example, to display the progress of an I/O operation.



## New **SslLibCallback**

---

**Purpose** Structure that represents a secure socket.

**Declared In** SslLib.h

**Prototype**

```
typedef struct {
    void *reserved;
    SslLibCallbackFunc callback;
    void *data;
    SslContext *ssl;
} SslLibCallback
```

**Fields** `callback`

A pointer to a callback function.

`data`

A pointer to data that’s passed to the function when it’s invoked.

`ssl`

The SSL context to which the callback applies, or NULL if the structure is owned by an SslLib object.

`reserved`

Reserved for future use.

## SSL Structures and Data Types

### SSL Structures

---

**Comments** To register a callback function, create a `SslLibCallback` structure, fill out the `callback` and `data` fields and pass the structure to one of the [`SslCallbackFunc`](#) functions. The structure is copied into the context, and the `ssl` field is properly set (in the copy).

It's the caller's responsibility to free the original `SslLibCallback` structure. The same `SslLibCallback` structure can be used to set more than one callback function.

For more information, see [`SslCallbackFunc`](#) in [Chapter 80, "SSL Functions."](#)



## SslSession

---

Structure that represents the current SSL session.

```
typedef struct SslSession_st
{
    UInt32 length;
    UInt16 version;
    unsigned char cipherSuite[2];
    unsigned char compression;
    unsigned char sessionId[33];
    unsigned char masterSecret[48];
    unsigned char time[8];
    unsigned char timeout[4];
    UInt16 certificateOffset;
    UInt16 extraData;
} SslSession;
```

A context's `SslSession` is populated when you call [`SslOpen`](#). You can retrieve a context's `SslSession` structure and modify it directly—particularly if you want to set the fields that are reserved for application use. To retrieve the `SslSession`, call the [`SslContextGet\_SslSession`](#) macro. Keep in mind that this gives you a pointer to the context's internal `SslSession` attribute. If you want to store the `SslSession`, you must copy the attribute's data.

The fields are:

length	The total size of the structure, in bytes.
version	Version number of the SSL protocol that's being used; 0 if the session is using cleartext.
cipherSuite	Cipher suite that's being used. See the <a href="#">CipherSuite</a> attribute.
compression	The name of the compression scheme that's being used.
sessionId	Session identification number. The ID value can be as many as 32 bytes long; the first byte gives the number of valid bytes.
masterSecret	The value of the “master secret” that was established during the SSL handshake.
time	Used to record the time that the session started, using a local time representation. The format can be anything that fits in eight bytes. Unused by the SSL library, an application can use this field as it wishes.
timeout	The number of seconds that the session should remain valid. The timeout field is also unused by the SSL library. Use it as you wish.
certificateOffset	Offset, in bytes, from the start of the structure to an <a href="#">SslExtendedItems</a> structure that contains the server's certificate. The field is provided for the application's convenience. The SSL library doesn't actually copy the certificate: If you want to cache the server's certificate, you have to retrieve the data through the <a href="#">PeerCert</a> attribute, extend the size of the SslSession structure, copy the certificate data, and then set the certificateOffset yourself.
extraData	Offset, in bytes, from the start of the structure to some extra data that the application keeps for itself. This is similar to the certificateOffset field: You have to allocate, copy, and mark the data yourself.

---

## SSL Structures and Data Types

### SSL Structures

---



## New SslSocket

The `SslSocket` structure contains information about the socket that's being used in the network connection. The structure is defined as:

```
typedef struct {
    NetSocketRef    socket;
    Int16          flags;
    UInt16         addrLen;
    Err            err;
    Int32          timeout;
    NetSocketAddrType  addr;
} SslSocket;
```

You can create your own `SslSocket` structure and add its data to a context through the [`SslContextSet\_IoStruct`](#) macro. When you do this, the data in your structure is copied into a structure that's held by the context. Note, however, that the `socket` field isn't copied. To set the `socket`, use [`SslContextSet\_Socket`](#).

The argument values that you pass to [`SslSend`](#) and [`SslReceive`](#) are used to modify the context's `SslSocket` structure. If, for example, you create an `SslSocket` with a particular `timeout` value, set it in a context, and then call [`SslSend`](#) with a different `timeout` value (passed as an argument), the latter `timeout` is recorded in the context's `SslSocket` structure, overriding your original setting.

To retrieve a pointer to the context's `SslSocket`, call [`SslContextGet\_IoStruct`](#).

The fields are:

socket	Socket identifier, as created and returned by <code>NetLibSocketOpen</code> . The SSL library doesn't open or connect the socket for you; all socket operations other than reading and writing data must be done through net library calls.  You never set the <code>socket</code> field directly; use the <a href="#"><u><code>SslContextSet_Socket</code></u></a> macro, instead.
flags	Options that control a socket's I/O operations. See <a href="#"><u>I/O Flags</u></a> in <a href="#"><u>Chapter 61, "Net Library,"</u></a> on page 1413 for a description of these options.
addr	The most recent address that the socket read from or wrote to.
addrLen	The length, in bytes, of the <code>addr</code> field.
err	The most recent socket error.
timeout	The socket's I/O operation timeout value, in system ticks.



## New **SslVerify**

Structure that's passed to a verify callback function to provide information about a certificate that's being verified. This happens when the verification process hits a snag, giving your app a chance to handle the problem. The callback is also invoked when the process has successfully completed.

## SSL Structures and Data Types

### SSL Structures

---

```
typedef struct SslVerify_st
{
    SslExtendedItems *certificate;
    SslExtendedItems *fieldItems;
    UInt32 field;
    SslExtendedItems *ex;
    UInt32 depth;
    UInt32 state;
} SslVerify;
```

The fields are:

---

certificate	Points to the certificate that's being verified.
fieldItems	fieldItems is an array of extended items; field is an index into fieldItems. The <a href="#">SslExtendedItems</a> that's located at fieldItems [ field ] contains the set of items that's causing the problem. (See below.)
field	
ex	If this is an extension error, the ex field contains the data element of the X509 extension that failed.
depth	This is the level of the certificate being processed, where 0 is the server's certificate, and higher numbers are certificates along the "certificate chain" that leads to a trusted root certificate.

---

The list of [SslExtendedItem](#) elements that the [SslExtendedItems](#) object located at fieldItems [ field ] contains depends on the error flavor:

<b>Flavor</b>	<b>Contents</b>
sslErrVerifyBadSignature	The server's certificate, which contains the public key entries.
sslErrVerifyNoTrustedRoot	NULL
sslErrVerifyNotAfter	sslExItemTypeX509, asn1FldX509NotAfter
sslErrVerifyNotBefore	sslExItemTypeX509, asn1FldX509NotBefore.
sslErrVerifyConstraintViolation	asn1ExItemTypeX509Ex (but see below)
sslErrVerifyUnknownCriticalExtension	asn1ExItemTypeX509Ex (but see below)
sslErrOk	NULL

The `asn1ExItemTypeX509Ex` item needs to be decoded in order to be used. The item contains the “object identifier” part of a certificate extension. The item in the next field (i.e. `fieldItems [ field+1 ]`) depends on the item’s `dataType` field: If it’s `asn1Boolean`, than the “+1” item contains the extension’s optional Boolean “this is a critical extension” field, otherwise it contains the extension data itself. If “+1” contains the Boolean, than “+2” is the data.

The SSL library attempts to interpret only critical extensions, so the critical field should always be present. If a critical extension isn’t understood, the certificate should be rejected.

## **SSL Structures and Data Types**

### *SSL Structures*

---

# SSL Attributes and Macros

---

As described in [Chapter 81, “SSL Structures and Data Types”](#), the `SslLib` and `SslContext` objects list and describe the properties, or **attributes**, of an SSL context.

There are a number of SSL attributes: Some provide general-purpose information that any SSL session needs, some are used only for specific protocols, some are provided for debugging purposes, and so on.

Associated with each attribute is a set of macros that get and set the attribute’s value.

This chapter lists and describes the SSL attributes, how they’re used, the values that they take (including defaults), and the macros that access them:

- “[SSL Macro Names](#)” explains how the macro names are formed, and how they correspond to the SSL attributes.
- “[SSL Attribute Data Types](#)” talks about the types of data an attribute can represent.
- “[SSL Macro Pseudo-Protocol](#)” provides a pseudo-protocol for the macros.
- “[SSL Attributes](#)” lists the attributes in alphabetical order.
- “[SSL Attribute Constants](#)” is a list of constants that represent the attributes in the attribute-setting functions (which you can use instead of the macros, although it isn’t recommended; the functions are explained below).

## SSL Macro Names

An SSL attribute is accessed through a set of (as many as) four macros: There’s a macro that lets you set an attribute’s value in an

## SSL Attributes and Macros

### *SSL Attribute Data Types*

---

[SslLib](#), another that sets the attribute's value in an [SslContext](#), and two others that retrieve attribute values from these objects.

The names of the four macros for a given attribute follow this pattern:

- `SslLibSet_Attribute` sets the attribute in an [SslLib](#).
- `SslLibGet_Attribute` gets the attribute from an [SslLib](#).
- `SslContextSet_Attribute` sets the attribute in an [SslContext](#).
- `SslContextGet_Attribute` gets the attribute from an [SslContext](#).

For example, the four macros that set and get the [Mode](#) attribute are:

- `SslLibSet_Mode`
- `SslLibGet_Mode`
- `SslContextSet_Mode`
- `SslContextGet_Mode`

Note that not all attributes require all four macros: Some attributes are read-only, while others apply only to [SslContext](#) objects.

The macros are covers for the eight attribute-setting functions described in [Chapter 80, “SSL Functions,”](#) on page 2135. It's recommended that you use the macros rather than the functions, but you should be aware of the functions' existence.

## SSL Attribute Data Types

An SSL attribute is either an integer or a pointer. For pointer attributes, the type of the pointed-to data depends on the attribute. The descriptions below explicitly state these data types. (The macro declarations in the `SslLibMac.h` file don't include the data types.)

When setting a pointer attribute, the pointed-to data is (almost always) copied into the [SslLib](#) or [SslContext](#). The original data needn't be preserved—the caller is free to destroy the original data when the pointer-setting macro returns. The exceptions to this rule are pointed out in the macro descriptions, below.

To retrieve a pointer attribute, you pass in the address of a pointer; the macro will reset the pointer to data that belongs to the [SslLib](#) or [SslContext](#). You mustn't free or otherwise modify this data.

The macros that set and get integer attributes are defined in terms of the [SslLibSetLong](#), [SslLibGetLong](#), [SslContextSetLong](#), and [SslContextGetLong](#) functions, described in [Chapter 80, “SSL Functions,”](#) on page 2135. The pointer-accessing macros are covers for an analogous set of functions ([SslLibSetPtr](#), *et al.*). You should rarely need to call these functions directly—they're meant to be called by the attribute macros.

## SSL Macro Pseudo-Protocol

This section provides pseudo-protocol for the attribute macros.



### **SslContextGet\_Attribute (integer version)**

**Purpose** Returns the value of an integer-valued attribute that's stored in an [SslContext](#) object. This is a cover for the [SslContextGetLong](#) function.

**Prototype** `Int32 SslContextSet_Attribute ( UInt16 libRef,  
SslContext *context )`

**Parameters** `-> libRef` (68k only) SSL library reference number.  
`-> context` The [SslContext](#) that owns the attribute.

**Result** Returns the attribute's value, or a negative-valued error code as described in [SslContextGetLong](#).

## SSL Attributes and Macros

### SSL Macro Pseudo-Protocol

---



#### New **SslContextGet\_Attribute (pointer version)**

**Purpose** Gets a pointer-valued attribute from an [SslContext](#) object. This is a cover for the [SslContextGetPtr](#) function.

**Prototype** `Int32 SslContextGet_Attribute ( UInt16 libRef,  
SslContext *context, void **data )`

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> context	The <a href="#">SslContext</a> that owns the attribute.
-> data	The macro resets <code>*data</code> so it points to the attribute's data. Unless otherwise noted, the pointed-to data belongs to the <a href="#">SslContext</a> and shouldn't be freed.

**Result** Returns 0 upon success; see [SslContextGetPtr](#) for error return values.



#### New **SslLibSet\_Attribute (integer version)**

**Purpose** Sets an integer-valued attribute in an [SslContext](#) object. This is a cover for the [SslLibSetLong](#) function.

**Prototype** `Err SslLibSet_Attribute ( UInt16 libRef,  
SslContext *context, Int32 value )`

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> context	The <a href="#">SslContext</a> that owns the attribute.
-> value	The value you want to set the attribute to.

**Result** Returns 0 upon success; see [SslLibSetLong](#) for error return values.



## New **SslContextSet\_Attribute (pointer version)**

**Purpose** Sets a pointer-valued attribute in an [SslContext](#) object. This is a cover for the [SslContextSetPtr](#) function.

**Prototype** Err SslContextSet\_Attribute ( UInt16 libRef,  
SslContext \*context, void \*data )

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> context	The <a href="#">SslContext</a> that owns the attribute.
-> data	A pointer to the data that you want to set.

**Result** Returns 0 upon success; see [SslContextSetPtr](#) for error return values.

**Comments** The data from data is copied into the [SslContext](#). You can free the original data after the macro returns.



## New **SslLibGet\_Attribute (integer version)**

**Purpose** Returns the value of an integer-valued attribute that's stored in an [SslLib](#) object. This is a cover for the [SslLibGetLong](#) function.

**Prototype** Int32 SslLibGet\_Attribute ( UInt16 libRef,  
SslLib \*lib )

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> lib	The <a href="#">SslLib</a> that owns the attribute.

**Result** Returns the attribute's value, or a negative-valued error code

## SSL Attributes and Macros

### *SSL Macro Pseudo-Protocol*

---



#### **New** **SslLibGet\_Attribute (pointer version)**

**Purpose** Gets a pointer-valued attribute from an [SslLib](#) object. This is a cover for the [SslLibGetPtr](#) function.

**Prototype** `Int32 SslLibGet_Attribute ( UInt16 libRef,  
SslLib *lib, void **data )`

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> lib	The <a href="#">SslLib</a> that owns the attribute.
-> data	The macro resets <code>*data</code> so it points to the attribute's data. Unless otherwise noted, the pointed-to data belongs to the <a href="#">SslLib</a> and shouldn't be freed.

**Result** Returns 0 upon success; see [SslLibGetPtr](#) for error return values.



#### **New** **SslLibSet\_Attribute (integer version)**

**Purpose** Sets an integer-valued attribute in an [SslLib](#) object. This is a cover for the [SslLibSetLong](#) function.

**Prototype** `Err SslLibSet_Attribute ( UInt16 libRef,  
SslLib *lib, Int32 value )`

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> lib	The <a href="#">SslLib</a> that owns the attribute.
-> value	The value you want to set the attribute to.

**Result** Returns 0 upon success; see [SslLibSetLong](#) for error return values.



## New **SslLibSet\_Attribute (pointer version)**

**Purpose** Sets a pointer-valued attribute in an [SslLib](#) object. This is a cover for the [SslLibSetPtr](#) function.

**Prototype** Err SslLibSet\_Attribute ( UInt16 libRef,  
SslLib \*lib, void \*data )

**Parameters**

-> libRef	(68k only) SSL library reference number.
-> lib	The <a href="#">SslLib</a> that owns the attribute.
-> data	A pointer to the data that you want to set.

**Result** Returns 0 upon success; see [SslLibSetPtr](#) for error return values.

**Comments** The data from data is copied into the [SslLib](#). You can free the original data after the macro returns.

## SSL Attributes



## New **AppInt32**

Convenient Int32 datum that the application can use for whatever purpose it wants—as an object ID, for example. Note that the value is *not* copied from an [SslLib](#) into the [SslContexts](#) that it spawns.

**type:** Int32

**macros:** SslLibSet\_AppInt32  
SslLibGet\_AppInt32  
SslContextSet\_AppInt32  
SslContextGet\_AppInt32

## SSL Attributes and Macros

### SSL Attributes

---



#### New AppPtr

Convenient pointer that the application can use for whatever purpose it wants.

**type:** (void \*)

The application retains ownership of the pointed-to data. The data is *not* copied into the [SslLib](#) or [SslContext](#).

**macros:** SslLibSet\_AppPtr  
SslLibGet\_AppPtr  
SslContextSet\_AppPtr  
SslContextGet\_AppPtr



#### New AutoFlush

Enables/disables automatic flushing of data after an [SslSend](#) or [SslWrite](#).

**type:** Int32

**values:** 0 : Autoflush is turned off.

1 : Autoflush is turned on.

**default:** 1 (autoflush on)

**macros:** SslLibSet\_AutoFlush  
SslLibGet\_AutoFlush  
SslContextSet\_AutoFlush  
SslContextGet\_AutoFlush

When enabled, [SslFlush](#) is automatically called after every call to [SslSend](#) and [SslWrite](#). If you have autoflush disabled, you have to call the [SslFlush](#) function yourself. You don't have to call it after *every* [SslSend](#) or [SslWrite](#), but you *do* have to call it before the [SslContext](#)'s write buffer fills up.

The write buffer's size is given by [WbufSize](#); the amount of data that's currently in the buffer is given by [WriteBufPending](#).



## New **BufferedReuse**

Allows the last message in an SSL handshake to be buffered. This only applies if [SessionReused](#) is true (non-zero).

**type:** Int32

**values:** 0: Don't buffer the last handshake message.  
non-zero: Buffer the last handshake message.

**macros:** SslContextGet\_BufferedReuse

As described in [SessionReused](#), the handshake involved in re-establishing a previous session can be a truncated version of a normal handshake. At the end of the truncated handshake, the SSL library sends a message to the server. The BufferReuse attribute, if enabled, will buffer the last message of the reused handshake instead of sending it over the network. The message will be sent when the application sends its first "real" data.

If you have BufferReuse enabled, you must make sure that the final handshake message actually gets sent before reading any incoming data.

There are security implications, here, in that a man-in-the-middle attack could only be detected once the first data bytes are read from the server. In other words, an attacker could have read all the bytes in the first "real data" message sent to the server. For this reason this attribute should not normally be used.

## SSL Attributes and Macros

### SSL Attributes

---



## CipherSuite

Represents the cipher suite that's currently in use in this context. [SslContext](#) only, read only.

**type:** (UInt8 \*)

**values:** Each suite is represented by a two-byte value.

0: No cipher suite is being used.

`sslCs_RSA_RC4_56_SHA1`: Secure Hash Algorithm-1, 56-bit

`sslCs_RSA_RC4_128_SHA1`: Secure Hash Algorithm-1, 128-bit.

`sslCs_RSA_RC4_40_MD5`: Rivest Message Digest 5, 40-bit.

`sslCs_RSA_RC4_128_MD5`: Rivest Message Digest 5, 128-bit.

**macros:** `SslContextGet_CipherSuite`

### See Also

[CipherSuites](#)



## CipherSuiteInfo

Returns information about the cipher suite or certificate that's currently being used. [SslContext](#) only, read only.

**type:** ([SslCipherSuiteInfo](#) \*)

The [SslCipherSuiteInfo](#) structure must be allocated before it's passed in. The caller retains ownership of the structure.

**values:** The structure is populated by the context. For information on what the structure contains, see [SslCipherSuiteInfo](#).

**macros:** `SslContextGet_CipherSuiteInfo`

To return *just* the cipher suite (without the surrounding information), get the [CipherSuite](#) attribute.



## CipherSuites

Contains a list of cipher suites that the client supports, in order of preference.

**type:** (UInt8 \*)

**values:** Each element in the list takes two bytes. The first two byte element give the sizes of the rest of the list, in bytes (i.e. the “size bytes” themselves aren’t counted in the measurement). The rest of the list is made up of the following (two-byte) constants:

```
sslCs_RSA_RC4_128_MD5  
sslCs_RSA_RC4_128_SHA1  
sslCs_RSA_RC4_56_SHA1  
sslCs_RSA_RC4_40_MD5
```

The data in the list is given in network byte order.

**default:** The default list contains {0x00, 0x08} followed by the suites listed above, in order. To restore the default list, pass a NULL pointer.

**macros:** SslLibSet\_CipherSuites  
SslLibGet\_CipherSuites  
SslContextSet\_CipherSuites  
SslContextGet\_CipherSuites

As an example, the following code creates a list of the strong encryption suites (only) and sets it into an [SslLib](#):

```
static UInt8 cipherSuites [] = {  
    0x00, 0x04,  
    sslCs_RSA_RC4_128_MD5,  
    sslCs_RSA_RC4_128_SHA1  
};  
  
SslLibSet_CipherSuites(lib, cipherSuites);
```

## SSL Attributes and Macros

### SSL Attributes

---

The cipher suites list is used when the client and server negotiate for an acceptable suite. The suite that's chosen is represented by the [CipherSuite](#) attribute.



## Compat

Allows compatibility with malformed or incorrect SSL messages sent from the server

**type:** Int32

**values:** 0 : No incompatibility allowed.

If non-zero, the value should be a combination of the following flags.

`sslCompatReuseCipherBug` : Allows servers to change cipher suites on session-reuse.

`sslCompatNetscapeCaDnBug` : Supports old versions of Netscape servers that encode "Distinguished Names" certificates incorrectly. (Note that this isn't currently an issue since the SSL library doesn't support client certificates.)

`sslCompat1RecordPerMessage` : Ensures that SSL messages aren't broken across records. (Some servers don't like to receive SSL messages separated into multiple SSL records.)

`sslCompatBigRecords` : Supports records larger than 16k bytes. (In SSLv3, record size is normally limited to 16k.)

`sslCompatAll` : Enables all the foregoing.

**default:** 0

**macros:** `SslLibSet_Compact`  
`SslLibGet_Compact`  
`SslContextSet_Compact`  
`SslContextGet_Compact`



## New **DontSendShutdown**

Avoid sending a shutdown message.

**type:** Int32

**values:** 0: Send the shutdown message.

non-zero: Don't send the shutdown message.

**default:** 0

**macros:** SslLibSet\_DontSendShutdown  
SslLibGet\_DontSendShutdown  
SslContextSet\_DontSendShutdown  
SslContextGet\_DontSendShutdown

When [SslClose](#) is called, the two SSL endpoints swap shutdown messages. You can suppress transmission of the local message by setting this attribute.

To ignore the remote shutdown message, set the [DontWaitForShutdown](#) attribute.



## New **DontWaitForShutdown**

Avoid waiting for reception of the server's shutdown message.

**type:** Int32

**values:** 0: Wait for the shutdown message.

non-zero: Don't wait the shutdown message.

**default:** 0

**macros:** SslLibSet\_DontWaitForShutdown  
SslLibGet\_DontWaitForShutdown  
SslContextSet\_DontWaitForShutdown  
SslContextGet\_DontWaitForShutdown

See [DontSendShutdown](#) for details.

## **SSL Attributes and Macros**

### *SSL Attributes*

---



#### **Error**

Stores the most recent fatal error code. [SslContext](#) only.

**type:** Int32

**values:** 0 : No error

non-zero: An error occurred somewhere up to here.

Note that SslErrIo errors are exempt (they're not considered to be fatal).

**default:** 0

**macros:** `SslContextSet_Error`  
`SslContextGet_Error`

After you read an Error value, you should set the attribute's value back to 0. The value is reset during an SSL Reset, but it otherwise holds the latest fatal error code.



#### **HsState**

Current SSL state. [SslContext](#) only, read only. Used primarily for debugging.

**type:** Int32

**values:** Defined by the SSL protocol.

**macros:** `SslContextGet_HsState`



#### **InfoCallback**

Identifies the callback function that's called at particular moments in an SSL transaction.

**type:** ([SslCallback](#) \*)

**default:** none

**macros:** SslLibSet\_InfoCallback  
SslLibGet\_InfoCallback  
SslContextSet\_InfoCallback  
SslContextGet\_InfoCallback

If you set the InfoCallback attribute, you must also set the [InfoInterest](#) attribute to register for the notifications that you want.

See the [SslCallbackFunc](#) description in [Chapter 80, “SSL Functions,”](#) on page 2135 for more information about the callback functions.

## SSL Attributes and Macros

### SSL Attributes

---



## New InfoInterest

Specifies the events for which the callback function identified by the [InfoCallback](#) attribute will be called.

**type:** Int32

**values:** The value is formed as a combination of the following (or 0 if you don't want your callback to be called):

`sslFlgInfoAlert`: The callback wants to know about alert notifications. In an alert, the function receives `sslArgInfoAlert` as its command argument.

`sslFlgInfoHandshake`: The callback wants to know about handshake notifications (`sslArgInfoHandshake`).

`sslFlgInfoIO`: The callback wants to be called before and after each [SslCallback](#) operation (`sslArgInfoReadBefore`, `sslArgInfoReadAfter`, `sslArgInfoWriteBefore`, `sslArgInfoWriteAfter`).

`sslFlgInfoCert`: The callback wants to know about certificate notifications (`sslArgInfoCert`).

**default:** 0

**macros:** `SslLibSet_InfoInterest`  
`SslLibGet_InfoInterest`  
`SslContextSet_InfoInterest`  
`SslContextGet_InfoInterest`

For these to be effective, you must set an info callback function (see [InfoCallback](#)).

For more information on how the callback functions work, and for detailed specifications of the command values referred to above, see the [SslCallbackFunc](#) description in [Chapter 80, “SSL Functions,”](#) on page 2135.



**New**

## **IoTimeout**

Amount of time that an I/O operation should wait for a response from the other side of then network connection before it gives up and returns an error. The timeout is measured in ticks.

**type:** Int32

**values:** [0, 0xffffffff].

**default:** 10 seconds

**macros:** `SslContextSet_IoTimeout`  
`SslContextGet_IoTimeout`

Individual SSL library functions can specify their own timeout values. In the absence of a timeout specification, the `IoTimeout` value is used.



**New**

## **IoFlags**

Socket I/O flags that are used by the data read and write functions. [SslContext](#) only.

**type:** Int32

**values:** See the “[I/O Flags](#)” on page 1421 for a list of these flags. `netIOFlagOutOfBand` and `netIOFlagPeek` don’t apply; they’re twiddled out of the value you supply.

**default:** none

**macros:** `SslContextSet_IoFlags`  
`SslContextGet_IoFlags`

## SSL Attributes and Macros

### *SSL Attributes*

---



**New**

### IoStruct

[SslSocket](#) structure that holds arguments that are passed to the underlying net library functions. [SslContext](#) only.

**type:** `(SslSocket *)`

**values:** See [SslSocket](#).

**default:** `(SslSocket *)NULL`

**macros:** `SslContextSet_IoStruct`  
`SslContextGet_IoStruct`

When you call `SslContextSet_IoStruct`, the socket field of the [SslSocket](#) that you pass in is ignored. Use the [Socket](#) attribute to set the socket.

The structure's other fields correspond to the arguments to the [SslSend](#) and [SslReceive](#) functions (which see for details). When you call one of these functions, the arguments that you pass are copied into the context's [SslSocket](#) structure. Thus, the [SslSocket](#) configuration that you set through `SslContextSet_IoStruct` may be updated when you call an I/O function.



**New**

### LastAlert

Most recent fatal error code. [SslContext](#) only, read only.

**type:** `Int32`

**values:** See the `sslAlertType` constants in `SslLibMac.h`.

**default:** 0

**macros:** `SslContextGet_LastAlert`

An alert is an error or status code that's sent by the server. Alerts can be fatal or non-fatal. Fatal alerts are in the form 0x02xx; non-fatal alerts are 0x01xx, as shown below:

```
Int32 alert;

SslContextGet_LastAlert( context, &alert);

if ( alert != 0 ) {
    if ( alert & 0x0f00 == 0x0200 )
        /* fatal */
    else if ( alert & 0x0f00 == 0x0100 )
        /* non-fatal */
    else
        /* undefined */
}
```

---

 **New**

## LastApi

Represents the most recently called SSL library function.  
[SslContext](#) only, read only.

**type:** Int32

**values:** `sslLastApiNone`: No SSL library function call in this context since the last SSL reset.

`sslLastApiRead`: The previous function was [SslOpen](#).

`sslLastApiRead`: The previous function was [SslRead](#), [SslPeek](#), or [SslReceive](#).

`sslLastApiWrite`: The previous function was [SslWrite](#) or [SslSend](#).

`sslLastApiFlush`: The previous function was [SslFlush](#).

`sslLastApiShutdown`: The previous function was [SslClose](#).

**macros:** `SslContextGet_LastApi`

## SSL Attributes and Macros

### *SSL Attributes*

---



**New**

### LastIO

Represents the nature of the previous I/O operation. [SslContext](#) only, read only.

**type:** Int32

**values:** `sslLastIoNone`: There is yet to be an I/O operation in this context since the last SSL reset.

`sslLastIoRead`: The previous I/O operation was a read.

`sslLastIoWrite`: The previous I/O operation was a write.

**default:** `sslLastIoNone`

**macros:** `SslContextGet_LastIO`

This attribute can be useful when you're determining the reason for an error.



**New**

### Mode

Turns SSL on and off.

**type:** Int32

**values:** `sslModeClear`: SSL is turned off.

`sslModeSsl`: SSL is turned on

`sslModeSslClient`: SSL is turned on, and this endpoint is a client.

`sslModeFlush`: Data-clearing flag. If present, the context's internal data buffers are cleared (the data is lost). This flag is applicable to [SslContexts](#) only.

**default:** sslModeSslClient

**macros:** SslLibSet\_Mode  
SslLibGet\_Mode  
SslContextSet\_Mode  
SslContextGet\_Mode

sslModeSsl and sslModeSslClient are essentially the same; the only difference is that the latter explicitly declares the object to represent the client side of the connection. In Palm OS 5, SSL endpoints are always clients.

When set to sslModeClear, SSL is bypassed. This lets you write your application with the SSL library API and still perform normal (non-SSL) data transfers. Your application will still get the advantage of the data buffering provided by the SSL I/O functions ([SslRead](#), [SslWrite](#), *et al.*).

You can use sslModeSsl as a mask to determine if SSL is on:

---

```
If (SslContextGet_Mode(ssl) & sslModeSsl)
    /* SSL protocol enabled */
else
    /* Using cleartext */
```

---

### The Mode Attribute in SslContexts

Setting an [SslContext](#)'s Mode (even if you set it to the status quo) causes an "SSL reset." This sets most of the other SSL attributes back to their default values, and puts the [SslContext](#) in the proper state to open a new SSL session. Note that an SSL reset doesn't clear SSL session information, thus allowing the previous session to be used in a new session with the same server. See [ReadStreaming](#) for details.

To force an SSL reset, do this:

---

```
SslContextSet_Mode(context, SslContextGet_Mode(context));
```

---

Switching from SSL-off to SSL-on initiates a new SSL handshake.

sslModeFlush is a flag that, when present, clears the internal read and write buffers. Any data in the buffers is lost. You would normally use this flag when you're reusing an SSL session.

## SSL Attributes and Macros

### *SSL Attributes*

---



#### New **PeerCert**

Structure that represents the certificate that was supplied by the server. [SslContext](#) only, read only.

**type:** ([SslExtendedItems](#) \*)

**values:** If a certificate is available, the context passes back a pointer to a structure. See [SslExtendedItems](#) for details.

**default:** ([SslExtendedItems](#) \*) NULL

**macros:** SslContextGet\_PeerCert



#### New **PeerCommonName**

Structure that contains information that you use to retrieve the server's common name from a previously retrieved certificate. [SslContext](#) only, read only.

**type:** ([SslExtendedItem](#) \*)

**values:** See the example, below.

**default:** ([SslExtendedItem](#) \*) NULL

**macros:** SslContextGet\_PeerCommonName

To use this attribute, you must also retrieve the server's certificate through [SslContextGet\\_PeerCert](#). You then perform some pointer juggling that's best explained through an example:

```
SslExtendedItems *cert;
SslExtendedItem *commonName;

/* The length of the common name */
Int16 length;

/* A pointer to the beginning of the common name. */
Int8 *bytes;

SslContextGet_PeerCert( ssl, &cert );
```

```
if ( cert != NULL ) {  
    SslContextGet_PeerCommonName( ssl, &commonName );  
  
    /* Get the common name length from the SslExtendedItem  
     * struct.  
     */  
    length = commonName->len;  
  
    /* The name itself is in the peer certificate, located  
     * at an offset (into the SslExtendedItems struct) as  
     * indicated by the offset field of the SslExtendedItem  
     * struct.  
     */  
    bytes = ((Int8 *)cert) + commonName->offset;  
  
    /* Now that we have a pointer to the name and the  
     * length of the name, we can compare it to the  
     * expected value.  
     */  
    StrNCompare ( bytes, expectedValue, length );  
}
```

---

If you're using SSL in an https context, for example, the client application should ensure that the common name contained in the server's certificate matches the requested URL.

 **New**

## ProtocolVersion

The SSL protocol that's being used

**type:** Int32

**values:** sslVersionSSLv3 only; if you set the protocol to some other value, SSL won't work.

**macros:** SslLibSet\_ProtocolVersion  
SslLibGet\_ProtocolVersion  
SslContextSet\_ProtocolVersion  
SslContextGet\_ProtocolVersion

## SSL Attributes and Macros

### *SSL Attributes*

---



**New**

## RbufSize

The size of the internal read buffer, in bytes.

**type:** Int32

**values:** [0, 16384]

**default:** 2048

**macros:** SslLibSet\_RbufSize  
SslLibGet\_RbufSize  
SslContextSet\_RbufSize  
SslContextGet\_RbufSize

The actual size of the read buffer may be different from that which you request: The buffer is automatically increased in size if the SSL protocol and/or decryption algorithm demand it. For example, if decryption can only operate on an entire record, and if it's passed a record that's longer than the read buffer size, the buffer will grow to accommodate the record.

The [ReadStreaming](#) attribute describes an advanced use of the read buffer that decreases data reading latency.



**New**

## ReadBufPending

Returns the number of bytes that are waiting to be read from the context's read buffer. The measurement includes in-coming SSL encryption data. [SslContext](#) only; read only.

**type:** Int32

**values:** [0, 16k]

**macros:** SslContextGet\_ReadBufPending

This attribute is provided for debugging purposes.



## New **ReadOutstanding**

Returns the number of bytes that remain to be read from the current SSL record. [SslContext](#) only; read only.

**type:** Int32

**values:** [0, 16k]

0 means the entire record has been read and verified.

**macros:** SslContextGet\_ReadOutstanding

See [ReadStreaming](#) for a justification of this attribute.



## New **ReadRecPending**

Returns the number of “real” bytes of data in the context’s read buffer. The measurement doesn’t include SSL encryption data. [SslContext](#) only; read only.

**type:** Int32

**values:** [0, 16k]

**macros:** SslContextGet\_ReadRecPending

If this attribute is 0, the next [SslRead](#) or [SslReceive](#) will cause a [NetLibReceive](#) invocation.



## New **ReadStreaming**

Allows partial record data to be read.

**type:** Int32

**values:** 0: Always wait for an entire record before reading.  
non-zero: Don’t wait for an entire record.

## SSL Attributes and Macros

### SSL Attributes

---

**default:** 0

**macros:** SslLibSet\_ReadStreaming  
SslLibGet\_ReadStreaming  
SslContextSet\_ReadStreaming  
SslContextGet\_ReadStreaming

If the network is running at a very low rate (such as a 300 baud modem), you may want to allow in-coming data to be returned to the application before a full record has been downloaded. Notice, however, that “stream reading” has an implication on security: Since you can’t verify the record until it has been fully read, you must be careful what you do with stream read data. You should never respond to in-coming data until an entire record has been read and verified. The [ReadOutstanding](#) attribute can be used to determine if a record has been fully read and verified.

The SSL library may reject stream reading requests. See the [Streaming](#) attribute for details.

To retrieve streaming data, use the [SslPeek](#) and [SslConsume](#) functions (rather than [SslRead](#) or [SslReceive](#)).



## SessionReused

---

Indicates whether the context is reusing a previous session.  
[SslContext](#) only, read only.

**type:** Int32

**values:** 0: No, this is a new session.

non-zero: Yes, this session is a continuation of a previous session.

**default:** 0

**macros:** SslContextGet\_SessionReused

SSL can re-establish a previously formed session if both ends of the network connection have a common notion of the session’s parameters (as represented by the [SslSession](#) attribute). Reusing a session lets the SSL protocol use a truncated (faster) handshake.



## New **Socket**

Specifies the net library socket that an [SslContext](#) will use in I/O operations. [SslContext](#) only.

**type:** [NetSocketRef](#) (Int16)

**default:** none

**macros:** [SslContextSet\\_Socket](#)  
[SslContextGet\\_Socket](#)

An SSL context can't perform any network operations until it's supplied with a valid, open, connected [NetSocketRef](#). The SSL library doesn't perform any Net library operations Creation, initialization, and destruction of the socket that the [NetSocketRef](#) represents is the application's responsibility.

This call is used to specify the net library socket that the [SslContext](#) should use to perform its network I/O operations. A [SslContext](#) is unable to perform any network operation until the application creates and supplies a suitable [NetSocketRef](#).

---

**IMPORTANT:** The SslLib library uses the [NetSocketRef](#) to send data and receive data only—it doesn't call any other net library functions except for [NetLibSend](#) and [NetLibReceive](#).  
**All socket creation, configuration, and shutdown operations must be performed by the application.**

---



## New **SslSession**

Structure that represents the current or impending SSL session. [SslContext](#) only.

**type:** ([SslSession](#) \*)

## SSL Attributes and Macros

### SSL Attributes

---

**values:** When a session is opened, the context populates the structure with information. See [SslSession](#) for details.

**macros:** `SslContextSet_SslSession`  
`SslContextGet_SslSession`

You can fine-tune an impending session by creating and populating an [SslSession](#) that you pass in through `SslContextSet_SslSession`. As with most pointer attributes, the structure is copied into the context. The next time you call [SslOpen](#), the embedded [SslSession](#) information is used to configure the session. The structure is then modified, if necessary, to reflect the actual state of the session.

Session information isn't cleared in an SSL reset. Maintaining the session info allows a session to be reused, as discussed in [SessionReused](#).



## SslVerify

---

Structure that's used during certificate verification. [SslContext](#) only, read only.

**type:** `(SslVerify *)`

**values:** See the example, below.

**default:** `(SslVerify *) NULL`

**macros:** `SslContextGet_SslVerify`

If certification runs into a problem that can't be resolved by the verify callback function (see [VerifyCallback](#)), or if you haven't registered a verify callback function, an error will be returned to your application, which can retrieve the [SslVerify](#) structure in an attempt to solve the conflict itself.

If your application determines that the error isn't fatal, it should clear the error (through [SslContextSet\\_Error](#)) and reinvoke the function that returned the error.

 **New**

## Streaming

Tells you if the context is stream reading. [SslContext](#) only; read only.

**type:** Int32

**values:** 0: The context isn't stream reading.

non-zero: The context is stream reading.

**macros:** SslContextGet\_Streaming

In some cases, the SSL library may decide that it doesn't want to do stream reading, regardless of the value of the [ReadStreaming](#) attribute. The value of this attribute tells you the truth.

 **New**

## VerifyCallback

Identifies the callback function that's used during certificate verification.

**type:** ([SslCallback](#) \*)

**default:** none

**macros:** SslLibSet\_VerifyCallback  
SslLibGet\_VerifyCallback  
SslContextSet\_VerifyCallback  
SslContextGet\_VerifyCallback

For more information, see the [SslCallbackFunc](#) description in [Chapter 80, “SSL Functions.”](#)

## SSL Attributes and Macros

### *SSL Attribute Constants*

---



**New**

#### **WbufSize**

The size of the internal write buffer, in bytes. Note that SSL protocol overhead takes about 30 bytes, so the “writable” portion of the write buffer will be slightly smaller than the value of `WbufSize`.

**type:** `Int32`

**values:** `[0, 16384]`

**default:** `1024`

**macros:** `SslLibSet_WbufSize`  
`SslLibGet_WbufSize`  
`SslContextSet_WbufSize`  
`SslContextGet_WbufSize`



**New**

#### **WriteBufPending**

Returns the number of bytes of data in the context’s write buffer (i.e. waiting to be sent to the remote endpoint). `SslContext` only; read only.

**type:** `Int32`

**values:** `[0, 16k]`

**macros:** `SslContextGet_WriteBufPending`

This attribute should be zero unless `AutoFlush` is disabled. You must explicitly flush (through `SslFlush`) before the value of `WriteBufPending` reaches the value of `WbufSize` (the maximum size of the write buffer).

## SSL Attribute Constants

As explained earlier, each of the attribute macros is a cover for one of the eight attribute-setting or -getting functions. (The functions are described in [Chapter 80, “SSL Functions,”](#) on page 2135.) Instead of

invoking the attribute macros, you can call the attribute functions, instead.

Unlike the macros, the functions don't correspond to specific attributes. To tell the function which attribute you want it to access, you pass in one of the attribute constants. These constants usually take the form

`sslAttrLibAttribute`

or

`sslAttrAttribute`

where the former represents the attribute in an [Ssllib](#) object, and the latter is the [SslContext](#) version (note the lack of a qualifying "Context" in the names of the [SslContext](#) macro constants). As examples:

- `sslAttrLibMode` represents the [Mode](#) attribute in an [Ssllib](#) object.
- `SslAttrMode` is the [Mode](#) attribute in an [SslContext](#).
- `sslAttrLibAutoFlush` and `sslAttrAutoFlush` signify the [AutoFlush](#) attribute in the [Ssllib](#) and [SslContext](#).

Note that there are a few exceptions to this formula.

Alphabetical lists of the attribute constants are given below, one list for [Ssllib](#) attributes, and a second for [SslContext](#). The constant names that don't conform to the formula, above, are noted.

### **Ssllib Attribute Constants**

`sslAttrLibAppInt`  
`sslAttrLibAppPtr`  
`sslAttrLibAutoFlush`  
`sslAttrLibBufferedReuse`  
`sslAttrLibCompat`  
`sslAttrLibDontSendShutdown`  
`sslAttrLibDontWaitForShutdown`  
`sslAttrLibInfoCallback`  
`sslAttrLibInfoInterest`  
`sslAttrLibMode`  
`sslAttrLibProtocolVersion`  
`sslAttrLibRbufSize`  
`sslAttrLibReadStreaming`  
`sslAttrLibSessionCallback`  
`sslAttrLibVerifyCallback`

## **SSL Attributes and Macros**

### *SSL Attribute Constants*

---

sslAttrLibWbufSize

#### **SslContext Attribute Constants**

sslAttrAppInt  
sslAttrAppPtr  
sslAttrAutoFlush  
sslAttrBufferedReuse  
sslAttrCertPeerCert -> [PeerCert](#)  
sslAttrCertPeerCommonName -> [PeerCommonName](#)  
sslAttrCertSslVerify -> [SslVerify](#)  
sslAttrCompat  
sslAttrCspCipherSuiteInfo -> [CipherSuiteInfo](#)  
sslAttrCspCipherSuites -> [CipherSuites](#)  
sslAttrCspCipherSuites -> [CipherSuite](#)  
sslAttrCspSslSession -> [SslSession](#)  
sslAttrDontSendShutdown  
sslAttrDontWaitForShutdown  
sslAttrError  
sslAttrHsState  
sslAttrInfoCallback  
sslAttrInfoInterest  
sslAttrIoFlags  
sslAttrIoSocket -> [Socket](#)  
sslAttrIoStruct  
sslAttrIoTimeout  
sslAttrLastAlert  
sslAttrLastApi  
sslAttrLastIo  
sslAttrMode  
sslAttrProtocolVersion  
sslAttrRbufSize  
sslAttrReadBufPending  
sslAttrReadOutstanding  
sslAttrReadRecPending  
sslAttrReadStreaming  
sslAttrSessionReused  
sslAttrStreaming  
sslAttrVerifyCallback  
sslAttrWbufSize  
sslAttrWriteBufPending

# SSL Error Codes

---

The SSL functions described in [Chapter 80, “SSL Functions,”](#) return `errNone` if successful, and non-zero otherwise. The table below lists and explains these “non-zero” values.

In addition to the errors listed below, the SSL functions also pass back errors that are returned by the underlying network library (NetLib) functions.

---

**WARNING!** The `sslErrOk` error code, declared in `SslLib.h` is does *not* indicate success. The error-returning SSL functions return `errNone` for success.

---

The errors are grouped under topical headers, and listed alphabetically therein. The groups are:

- [SSL Function Protocol Errors](#) indicate a malformed function call, or other non-SSL dilemma.
- [SSL Alerts](#) are errors that your application may need to explore to find out what really happened.
- [SSL Handshake Errors](#) are returned while SSL handshake messages are being passed between the client and the server.
- [SSL Cryptography Errors](#) indicate that a cryptographic operation has failed.
- [SSL Illegal Message Errors](#) are returned when SSL receives an unexpected message type.
- [SSL Certificate Errors](#) describe the different failures during certificate verification.

## SSL Function Protocol Errors

`sslErrBadArgument` A function argument was (generally) invalid.

## SSL Error Codes

### *SSL Alerts*

---

<code>sslErrBadLength</code>	A buffer length argument to a data-reading or -writing function was invalid.
<code>sslErrBadOption</code>	This means the same as <code>sslErrBadArgument</code> .
<code>sslErrBufferTooSmall</code>	A buffer supplied to this function wasn't big enough for the data that the functions wanted to stuff into it.
<code>sslErrEof</code>	You attempted to read or write a socket that isn't open.
<code>sslErrFailed</code>	General, unspecified error.
<code>sslErrNullArg</code>	You passed in a NULL argument. This is returned, for example, if you pass in a NULL pointer where a valid <a href="#">SslLib</a> or <a href="#">SslContext</a> is expected.
<code>sslErrOutOfMemory</code>	Not enough memory to allocate the resources that are needed by this function.

## SSL Alerts

<code>sslErrCbAbort</code>	Returned by a callback function to indicate that the caller should exit. See <a href="#">SslCallbackFunc</a> in <a href="#">Chapter 80, “SSL Functions.”</a>
<code>sslErrIO</code>	Indicates that an underlying NetLib function has returned a non-fatal error, such as a timeout. When you get this error, you should try to determine what the exact error is—you usually start by looking at the context's <a href="#">Compat</a> attribute. After you clear the error, you can try to call the function again.
<code>sslErrFatalAlert</code>	An SSL fatal alert was received. To handle the alert, see the <a href="#">LastAlert</a> attribute in <a href="#">Chapter 82, “SSL Attributes and Macros.”</a>

## SSL Handshake Errors

`sslErrBadPeerFinished`

The final check of the SSL handshake failed. This means that there was a problem establishing a shared secret value. It could be caused by the server using a certificate that does not match its private key.

`sslErrExtraHandshakeData`

An SSL handshake message contained data that shouldn't have been there.

`sslErrHandshakeEncoding`

An error occurred while an SSL handshake message (from the server) was being decoded.

`sslErrHandshakeProtocol`

An error occurred while processing a decoded SSL handshake message.

`sslErrReadAppData`

A handshake message was expected, but application data was read.

## SSL Cryptography Errors

`sslErrCsp` General cryptography error.

`sslErrDivByZero`

Math error. This can happen if a certificate has an invalid public key.

`sslErrNoModInverse`

Another math error. See above.

`sslErrNoRandom` There was a problem with the random number source.

## **SSL Error Codes**

### *SSL Illegal Message Errors*

---

## **SSL Illegal Message Errors**

`sslErrBadSignature`

An invalid signature was found on an ephemeral Cipher Suite message.

`sslErrWrongMessage`

An invalid or inappropriate SSL message was received.

`sslErrUnexpectedRecord`

The wrong type of record was received.

`sslErrRecordError`

An invalid record was received by the [SslContext](#).

## **SSL Certificate Errors**

`sslErrBadDecode`

Something went wrong while decoding values during certificate verification.

`sslErrCert`

General certificate error.

`sslErrCertDecodeError`

The server's certificate could not be decoded.

`sslErrUnsupportedCertType`

The server sent a certificate that isn't supported (it contains a public key that can't be decoded).

`sslErrUnsupportedSignatureType`

The server sent a certificate that has an unrecognized signature type.

`sslErrVerifyBadSignature`

The certificate's signature is invalid

`sslErrVerifyConstraintViolation`

The certificate violates an X509 extension.

`sslErrVerifyNotAfter`

The certificate has expired.

`sslErrVerifyNotBefore`

The certificate is too early (the timestamp window is in the future).

`sslErrVerifyNoTrustedRoot`

A trusted certificate store (necessary for certificate verification) couldn't be found.

`sslErrVerifyUnknownCriticalExtension`

An X509 extension (that's marked as "critical") isn't understood by the certificate verification routines.

## **SSL Error Codes**

### *SSL Certificate Errors*

---

# SMS Exchange Library

---

This chapter describes the SMS Exchange Library API declared in the header file `SmsLib.h`. It discusses the following topics:

- [SMS Exchange Library Data Structures](#)
- [SMS Exchange Library Constants](#)

You interact with the SMS Exchange Library using the Exchange Manager APIs described in [Chapter 57, “Exchange Manager”](#) on page 1297 of this book. For further information on using Exchange Manager, see [“Object Exchange”](#) on page 1 of the *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## SMS Exchange Library Data Structures

### **SmsParamsType**

The `SmsParamsType` structure identifies information specific to the SMS Exchange Library. The `socketRef` field of the [`ExgSocketType`](#) structure is set to this structure when you send or receive data using the SMS Exchange Library. You only need to create this structure and assign it to the `socketRef` field if you have an SMS message to send and want to use non-default values for some of the fields; otherwise, the SMS Exchange Library creates this structure for you and provides default values.

```
typedef struct SmsParamsTag
{
    UInt32 creator;
    UInt16 smsID;
    Char *extension;
    Char *mimeTypes;
    UInt32 appCreator;
```

## SMS Exchange Library

### *SMS Exchange Library Data Structures*

---

```
    UInt8 dataCodingScheme;
    UInt8 networkType;
    UInt8 dataType;
    UInt16 nbsDestPort;
    UInt16 nbsSrcPort;
    union
    {
        SmsSendParamsType send;
        SmsReceiveParamsType receive;
        SmsReportParamsType report;
    } data;
} SmsParamsType, *SmsParamsPtr;
```

#### Field Descriptions

creator	Creator ID of the SMS Exchange Library. Always set this to <code>sysFileCSmsLib</code> .
smsID	The ID of the message that was sent. Do not set this field directly; the SMS Exchange Library should set it.
extension	If the SMS message has an attachment, this field specifies the attachment name. Do not set this field directly; the SMS Exchange Library sets it if necessary. See the <code>appCreator</code> field description for details.
mimeTypes	If the SMS message has an attachment, this field specifies the MIME type of the attachment. Do not set this field directly; the SMS Exchange Library sets it if necessary. See the <code>appCreator</code> field description for details.

`appCreator`

The creator ID of the target application for the attachment to the SMS message. Do not set this field directly; the SMS Exchange Library sets it if necessary.

When the SMS Exchange Library receives a message with an attachment, it unwraps the message and attempts to deliver the attachment directly to an application that is registered to receive it. If no application is registered to receive unwrapped attachments of that type, the SMS Exchange Library sends the entire SMS message, and it sets the `extension`, `mimeTypes`, and `appCreator` fields in this structure. The SMS application can use this information to have the Exchange Manager deliver the attachment to the appropriate application using the Local Exchange Library.

`dataCodingScheme`

The data encoding scheme that the message uses. See [SMS Data Coding Scheme Constants](#).

`networkType`

Indicates the type of advanced parameters. See [SMS Network Type Constants](#).

`dataType`

Identifies the type of message being received, such as multipart or return receipt. See [SMS Message Type Constants](#).

`nbsDestPort`

The Narrow Band Socket (NBS) port on which you want the data sent. The SMS Exchange Library sets this for you if you leave it blank. When data is being received, this field is set to the NBS port on which the data was received.

## SMS Exchange Library

### *SMS Exchange Library Data Structures*

---

nbsSrcPort	The NBS port on which you want the data sent. You should set this field to the same value as nbsDestPort. If you leave it blank, the SMS Exchange Library provides a value for you.
data	The data being sent or received. See <a href="#">SmsReceiveParamsType</a> , <a href="#">SmsReportParamsType</a> , and <a href="#">SMSSendParamsType</a> .

**Compatibility** This structure is only defined if [4.0 New Feature Set](#) is present.

## SmsPrefType

The SmsPrefType structure defines the SMS Exchange Library preferences for sending and receiving SMS messages. Applications can use the [ExgControl](#) function to get, set, or display these preferences to the user.

```
typedef struct SmsPrefTag
{
    UInt32 validity;
    UInt16 warnOver;
    Boolean leave;
    Boolean report;
    Boolean autoSMSC;
    Char smscNumber[kSmsMaxPhoneSize];
} SmsPrefType, *SmsPrefPtr;
```

### Field Descriptions

validity	The number of seconds before the message expires. If the message cannot be delivered to the recipient, the service center repeatedly attempts to deliver the message until it expires. The default is one hour.
warnOver	The number of parts a user can send without confirmation. If the user attempts to send a message with more than this number of parts, an alert is displayed, and the user can choose to send the message anyway. The default is 3 parts. (If the user attempts to send a message with more than 3 parts, an alert is displayed.)
leave	If <code>true</code> , any incoming messages retrieved from a phone remain on the phone as well. If <code>false</code> , the messages are deleted from the phone's inbox.
report	If <code>true</code> , the user receives confirmation that an SMS message was delivered.
autoSMSC	If <code>true</code> , don't use the value stored in the <code>smscNumber</code> field.
smscNumber	The message center to be used. If <code>NULL</code> or the empty string, the SMS message center set by the phone is used.

**Compatibility** This structure is only defined if [4.0 New Feature Set](#) is present.

## SmsReceiveCDMAParamsType

The [SmsReceiveParamsType](#) includes an `SmsReceiveCDMAParamsType` structure for CDMA messages.

```
typedef struct SmsReceiveCDMAParamsTag
{
    UInt8 messageType;
    TelSmsDateTimeType validityPeriod;
    UInt8 priority;
    UInt8     privacy;
```

## SMS Exchange Library

### SMS Exchange Library Data Structures

---

```
Boolean alertOnDeliveryRequest;
Boolean manualAckRequest;
UInt8 voiceMessageNumber;
UInt8 languageIndicator;
Char * callbackNumberAddress;
} SmsReceiveCDMAParamsType,
*SmsReceiveCDMAParamsPtr;
```

#### Field Descriptions

messageType	The type of the message. This is one of the <a href="#">SMS Message Type Constants</a> constants defined in <code>TelephonyMgr.h</code> .
validityPeriod	The amount of time for which the message is valid. See <a href="#">TelSmsDateTimeType</a> . The default is set according to the SMS preferences.
priority	The message priority. This must be one of the <a href="#">SMS Message Urgency Constants</a> defined in <code>TelephonyMgr.h</code> .
privacy	The privacy type of the message. This must be one of the <a href="#">SMS Message Privacy Constants</a> defined in <code>TelephonyMgr.h</code> .
alertOnDeliveryRequest	true if the user is to be alerted upon delivery of this message, and false if not.
manualAckRequest	true if a confirmation is requested from the recipient, and false if not.
voiceMessageNumber	The number of new messages in your voice mail.

languageIndicator	Reserved for future use.
callbackNumberAddress	The callback number to which confirmations are to be sent.

**Compatibility** This structure is only defined if [4.0 New Feature Set](#) is present.

## SmsReceiveGSMPParamsType

The [SmsReceiveParamsType](#) includes an SmsReceiveGSMPParamsType structure for GSM messages.

```
typedef struct SmsReceiveGSMPParamsTag
{
    UInt16 protocolId;
    Char *serviceCenterNumber;
    Boolean replyPath;
} SmsReceiveGSMPParamsType,
*SmsReceiveGSMPParamsPtr;
```

### Field Descriptions

protocolId	Reserved for future use.
serviceCenterNumber	The SMS service center that must be used to send a reply. If NULL, the service center specified in the preferences is used.
replyPath	If true, replies must be made through the SMS service center specified by serviceCenterNumber.

**Compatibility** This structure is only defined if [4.0 New Feature Set](#) is present.

## SmsReceiveParamsType

The SmsReceiveParamsType structure is used as the data field for the [SmsParamsType](#) structure when the SMS Exchange Library

## SMS Exchange Library

### SMS Exchange Library Data Structures

---

has received data. The SMS Exchange Library always supplies the values for these fields.

```
typedef struct SmsReceiveParamsTag
{
    UInt32 timeStamp;
    Char *originatingAddress;
    UInt8 leaveOnPhone:1;
    UInt8 forceSlotMode:1;
    UInt8 reserved:6;
    UInt16 index;
    Boolean otherToReceive;
    Boolean reportDeliveryIndicator;
    union
    {
        SmsReceiveGSMParamsType gsm;
        SmsReceiveCDMAParamsType cdma;
        SmsReceiveTDMAParamsType tdma;
    } protocol;
} SmsReceiveParamsType, *SmsReceiveParamsPtr;
```

#### Field Descriptions

timeStamp	The time at which the message was delivered, given as the number of seconds since January 1, 1904.
originatingAddress	The number from which the message was received.
leaveOnPhone	If true, messages received on the phone are not deleted from the phone's inbox. If not specified, this is set according to the system preferences.
forceSlotMode	If true, use slot mode parsing. If false, use block mode parsing. The default is block mode.
reserved	Reserved for future use.

index	Location where the message is stored on the mobile phone.
otherToReceive	If true, there are more messages to be received from the service center.
reportDeliveryIndicator	If true, the sender has requested confirmation. The recipient of the message does not send the confirmation; the SMS service center does.
protocol	Values specific to the protocol used to send the message. Currently, only GSM is supported.

**Compatibility** This structure is only defined if [4.0 New Feature Set](#) is present.

## SmsReceiveTDMAParamsType

The [SmsReceiveParamsType](#) includes an SmsReceiveTDMAParamsType structure for TDMA messages. This structure is currently the same as the [SmsReceiveCDMAParamsType](#) structure.

```
typedef SmsReceiveCDMAParamsType  
        SmsReceiveTDMAParamsType,  
        *SmsReceiveTDMAParamsPtr;
```

**Compatibility** This structure is only defined if [4.0 New Feature Set](#) is present.

## SmsReportParamsType

The SMSReportParamsType structure is used as the data field for the [SmsParamsType](#) structure when the SMS Exchange Library has received a delivery confirmation. The SMS Exchange Library always sets the values for these fields.

```
typedef struct SmsReportParamsTag  
{  
    UInt32 timeStamp;
```

## SMS Exchange Library

### SMS Exchange Library Data Structures

---

```
    UInt16 index;
    UInt8 reportType;
    UInt8 report;
    Char* originatingAddress;
} SmsReportParamsType, *SmsReportParamsPtr;
```

#### Field Descriptions

timeStamp	The date and time at which the message was delivered, given as the number of seconds since January 1, 1904.
index	Location where the message is stored on the mobile phone.
reportType	One of the Delivery Report Type constants defined in <code>TelephonyMgr.h</code> .
report	One of the Delivery Status Report constants defined in <code>TelephonyMgr.h</code> .
originatingAddress	Phone number to which the message was sent.

**Compatibility** This structure is only defined if [4.0 New Feature Set](#) is present.

## SmsSendCDMAParamsType

The [SMSSendParamsType](#) includes an `SmsSendCDMAParamsType` structure for CDMA messages.

```
typedef struct SmsSendCDMAParamsTag
{
    UInt8 messageType;
    TelSmsDateTimeType deferredDate;
    UInt8 priority;
    UInt8 privacy;
    UInt8 alertOnDelivery:1;
    UInt8 manualAckRequest:1;
    UInt8 reserved:6;
    Char* callbackNumber;
} SmsSendCDMAParamsType, *SmsSendCDMAParamsPtr;
```

## Field Descriptions

messageType	The type of the message. This is one of the <a href="#">SMS Message Type Constants</a> constants defined in <code>TelephonyMgr.h</code> .
deferredDate	Not used.
priority	The message priority. This must be one of the <a href="#">SMS Message Urgency Constants</a> .
privacy	The privacy type of the message. This must be one of the <a href="#">SMS Message Privacy Constants</a> .
alertOnDelivery	true if the user is to be alerted upon delivery of this message, and false if not.
manualAckRequest	true if a confirmation is requested from the recipient, and false if not.
reserved	Reserved for future use.
callbackNumber	Number to which the confirmation should be sent.

**Compatibility** This structure is only defined if [4.0 New Feature Set](#) is present.

## SmsSendGSMPParamsType

The [SMSParamsType](#) includes an `SmsSendGSMPParamsType` structure for GSM messages.

```
typedef struct SmsSendGSMPParamsTag
{
    UInt16 protocolId;
    Char *serviceCenterNumber;
    Boolean rejectDuplicated;
    Boolean replyPath;
} SmsSendGSMPParamsType, *SmsSendGSMPParamsPtr;
```

## SMS Exchange Library

### *SMS Exchange Library Data Structures*

---

#### Field Descriptions

protocolId	Reserved for future use.
serviceCenterNumber	The message center to be used. If not specified, the service center is set according to the system preferences.
rejectDuplicated	If true, the service center rejects messages that have the same message ID, destination address, and originating address as a previously submitted message.
replyPath	If true, the service center that delivers the message is requested to provide information about itself to the recipient so that replies are made through the same service center.

**Compatibility** This structure is only defined if [4.0 New Feature Set](#) is present.

### SMSSendParamsType

The SMSSendParamsType structure is used as the data field for the [SmsParamsType](#) structure when the SMS Exchange Library is sending data.

```
typedef struct SmsSendParamsTag
{
    TelSmsDateTimeType validityPeriod;
    Char *destinationAddress;
    UInt8 networkDeliveryRequested:1;
    UInt8 ignoreDefaultValue:1;
    UInt8 reserved:6;
    UInt16 partCount;
    UInt16 lastPart;
    UInt8 converter;
    union
    {
        SmsSendGSMPParamsType gsm;
        SmsSendCDMAParamsType cdma;
    }
}
```

```
    SmsSendTDMAParamsType tdma;  
} protocol;  
} SmsSendParamsType, *SmsSendParamsPtr;
```

### Field Descriptions

validityPeriod	The amount of time for which the message is valid. See <a href="#">TelSmsDateTimeType</a> . The default is set according to the SMS preferences.
destinationAddress	A buffer that contains the phone number of the message recipient. If no phone number is supplied, the user is prompted for the phone number.
networkDeliveryRequested	If true, the SMS service center sends a delivery confirmation. The default is set according to the SMS preferences. The SMS Exchange Library disables this field for multipart messages.
ignoreDefaultValue	If false, the validity period, network delivery requested, and SMS center specified in the preferences are used regardless of the values supplied in this structure. If true, the values supplied in this structure are used.
reserved	Reserved for future use.
partCount	The number of parts in the message. 0 means that the message is not a multipart message. If NBS is used to send the message, it determines the number of parts.

lastPart	The last part of a multipart message that was successfully sent.
converter	The header added to the data to specify how it is converted.
protocol	Data specific to the protocol used to send the message. Currently, only GSM is supported.

**Compatibility** This structure is only defined if [4.0 New Feature Set](#) is present.

## **SmsSendTDMAParamsType**

The [SmsSendParamsType](#) includes an SmsSendTDMAParamsType structure for TDMA messages. This structure is currently the same as the [SmsSendCDMAParamsType](#) structure.

```
typedef SmsSendCDMAParamsType  
        SmsSendTDMAParamsType,  
        *SmsSendTDMAParamsPtr;
```

**Compatibility** This structure is only defined if [4.0 New Feature Set](#) is present.

# **SMS Exchange Library Constants**

## **SMS Control Constants**

The SMS control constants are passed as the operation parameter to the [ExgControl](#) function. The ExgControl function is a way to communicate directly with the SMS Exchange Library. The following table lists the operation constant, the type of data that should be passed as the valueP parameter to [ExgControl](#), and what the SMS Exchange Library does in response.

**Table 84.1 ExgControl operations for SMS library**

<b>Operation</b>	<b>value Data Type</b>	<b>Description</b>
exgLibSmsPrefGetOp	<a href="#">SmsPrefType</a>	Returns a pointer to the SMS Exchange Libraries preferences in valueP, creating the preferences and setting them to the default values if they do not exist.
exgLibSmsPrefGetDefaultOp	<a href="#">SmsPrefType</a>	Returns the default values for the SMS Exchange Library preferences.
exgLibSmsPrefSetOp	<a href="#">SmsPrefType</a>	Sets the SMS Exchange Library preferences to the values passed in valueP.
exgLibSmsPrefDisplayOp	kSmsNetworkAuto or kSmsNetworkGSM. Input only.	Display a form that allows the user to set the SMS preferences.
exgLibSmsIncompleteGetCountOp	UInt16. Output only.	Get the number of incomplete messages currently stored in the SMS Exchange Library. The library stores message parts as it receives them. When it has received all of the parts, it reassembles the message and delivers it. This operation tells how many messages are currently under assembly.
exgLibSmsIncompleteDeleteOp	UInt16. Input only.	Delete the incomplete message with the ID passed in valueP. Pass -1 to delete all incomplete messages.

**Compatibility** These constants are only defined if [4.0 New Feature Set](#) is present.

## SMS Data Coding Scheme Constants

The SMS data coding scheme constants describe the coding scheme used for SMS data. These values are used as the `dataCodingScheme` parameter of the [SmsParamsType](#) structure.

Constant	Value	Description
<code>kSmsRowDataEncoding</code>	0	8-bit encoding scheme. This is the default.
<code>kSmsTextEncoding</code>	1	7-bit encoding scheme.

**Compatibility** These constants are only defined if [4.0 New Feature Set](#) is present.

## SMS Network Type Constants

The SMS network type constants identify the type of network being used for SMS messages. Currently, only the GSM network is supported.

Constant	Value	Description
<code>kSmsNetworkAuto</code>	-1	The network is set by the phone. This is the default.
<code>kSmsNetworkCDMA</code>	<code>kTelNwkCDMA</code>	A CDMA network. Currently not supported.
<code>kSmsNetworkGSM</code>	<code>kTelNwkGSM</code>	A GSM network.
<code>kSmsNetworkTDMA</code>	<code>kTelNwkTDMA</code>	A TDMA network. Currently not supported.
<code>kSmsNetworkPDC</code>	<code>kTelNwkPDC</code>	A PDC network. Currently not supported.

**Compatibility** These constants are only defined if [4.0 New Feature Set](#) is present.

## SMS Message Type Constants

The SMS message type constants identify the type of message being sent. They are used as the `dataType` field of the [SmsParamsType](#) structure.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
kSmsMessageType	0	Standard SMS message of no more than 160 bytes. This is the default.
kSmsIncompleteType	1	A part of a multipart SMS message.
kSmsReportType	2	A confirmation, indicating that an SMS message was successfully sent.

**Compatibility** These constants are only defined if [4.0 New Feature Set](#) is present.

## SMS Converter Constants

The SMS converter constants identify the header information added to an SMS message.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
kSmsNBSConverter	0	An NBS header is added to the message. This is the default.
kSmsNoConverter	1	No header is added to the message.

**Compatibility** These constants are only defined if [4.0 New Feature Set](#) is present.

## **SMS Exchange Library**

### *SMS Exchange Library Constants*

---

# Personal Data Interchange Library

---

This chapter provides reference material for the Personal Data Interchange (PDI) library, which provides tools for reading and writing vObjects, including vCards and vCals. This chapter discusses the following topics:

- [PDI Library Data Structures](#)
- [PDI Library Constants](#)
- [PDI Library Functions](#)

The header file `PdiLib.h` declares the Personal Data Interchange library API. The header file `PdiConst.h` declares the constants that you use with the PDI library.

For information about how to use the functions and constants described in this chapter, see [Chapter 3, “Personal Data Interchange,”](#) in *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## PDI Library Data Structures

This section describes the data structures used with the PDI library functions.

### **PdiDictionary**

The `PdiDictionary` type is a simple typedef that represents an internal, binary object.

```
typedef UInt8 PdiDictionary;
```

## PdiReaderType

The PdiReaderType data structure represents a PDI reader object, which you use to read data from an input stream.

```
typedef struct _PdiReader {
    Err          error;
    UInt8        encoding;
    CharEncodingType charset;
    UInt16       written;
    UInt8        fieldNum;
    UInt16       property;
    UInt16       propertyValueType;
    UInt16       parameter;
    UInt32      parameterPairs [kPDI_ENTRIES_NUMBER];
    UInt16       customFieldNumber;
    void         *appData;
    UInt16       pdiRefNum;
    UInt16       events;
    Char         *groupName;
    Char         *propertyName;
    Char         *parameterName;
    Char         *parameterValue;
    Char         *propertyValue;
    PdiDictionary *dictionary [2];
} PdiReaderType
```

### Field Descriptions

error	The most recent error.
encoding	The type of encoding for the property value.
charset	The character set of the property value.
written	The number of characters that have currently been written to the buffer.
fieldNum	The current field number.

property	The ID of the current property.
propertyValueType	The value type of the current property value.
parameter	The ID of the most recently parsed parameter name.
parameterPairs	An integer array with bits set for each parameter value that has been parsed for the current property value.

---

**NOTE:** You must use the `PdiParameterPairTest` macro to access this field.

---

customFieldNumber	The number of the custom field parsed by the reader for the current property. Custom fields are used in the Palm™ Address Book.
appData	Application-dependent data field.
pdiRefNum	The library reference number associated with this reader.
events	The mask of events handled by the reader in its most recent operation. This is a combination of some number of the event constants described in <a href="#">Reader Event Constants</a> .
groupName	The group name for the current property.
propertyName	The name of the current property.
parameterName	The name of the current parameter.
parameterValue	The value of the current parameter.

## Personal Data Interchange Library

### PDI Library Data Structures

---

propertyValue	The current property value string.
dictionary	An array of two dictionary pointers. The dictionary [0] entry is the main dictionary built into the PDI library, and the dictionary [1] entry is an optional, custom dictionary associated with the reader object with a call to <a href="#"><u>PdiDefineReaderDictionary</u></a> .

## PdiWriterType

The PdiWriterType data structure represents a PDI writer object, which you use to write data to an output stream.

```
typedef struct _PdiWriter {
    void             *appData;
    UInt16           pdiRefNum;
    UInt16           encoding;
    CharEncodingType charset;
    Err              error;
    PdiDictionary   *dictionary[2];
} PdiWriterType
```

### Field Descriptions

appData	Application-dependent data field.
pdiRefNum	The library reference number associated with this reader.
encoding	The type of encoding for the property value.
charset	The character set of the property value.

error	The most recent error.
dictionary	An array of two dictionary pointers. The dictionary [0] entry is the main dictionary built into the PDI library, and the dictionary [1] entry is an optional, custom dictionary associated with the reader object with a call to <a href="#"><u>PdiDefineReaderDictionary</u></a> .

## PDI Library Constants

This section describes the constants used in the PDI library, which include the following constant types:

- [Buffer Management Constants](#)
- [Encoding Type Constants](#)
- [Error Code Constants](#)
- [Parameter Name Constants](#)
- [Parameter Value Constants](#)
- [Property Name Constants](#)
- [Property Type Constants](#)
- [Property Value Field Constants](#)
- [Property Value Format Constants](#)
- [Reader and Writer Options Constants](#)
- [Reader Event Constants](#)
- [Value Type Constants](#)

### Buffer Management Constants

You use the buffer management constants to determine how buffers are managed in the PDI reader.

## Personal Data Interchange Library

### PDI Library Constants

---

Constant	Value	Description
kPdiResizableBuffer	0xFFFF	Indicates that the buffer can be automatically resized by the PDI library.
kPdiDefaultBufferSize	0x3FFF	The default maximum buffer size, in bytes. You can change the maximum size of a reader's buffer by calling the <a href="#">PdiDefineResizing</a> function.
kPdiDefaultBufferDeltaSize	0x0010	The default number of bytes by which the input buffer is grown when the PDI library performs automatic resizing. You can change the delta amount of a reader's buffer by calling <a href="#">PdiDefineResizing</a> function.

## Encoding Type Constants

You use the encoding type constants to specify the type of encoding used in a vObject reader or writer.

Constant	Value	Description
kPdiASCIIEncoding	0	The vObject is not encoded.
kPdiQPEncoding	kPdiPAV_ENCODING_QUOTED_PRINTABLE	The vObject uses the quoted printable encoding.
kPdiB64Encoding	kPdiPAV_ENCODING_BASE64	The vObject uses Base 64 encoding. The writer outputs "ENCODING=BASE64."
kPdiBEncoding	kPdiPAV_ENCODING_B	The vObject uses Base 64 encoding. This is the same as the kPdiB64Encoding value, except that the PDI writer outputs "ENCODING=B."
		This encoding is used with the vCard 3.0 standard.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
kPdiEscapeEncoding	0x8000	The vObject uses escapes for special characters.
kPdiNoEncoding	0x8001	The PDI writer does not encode the vObject value.

---

## Error Code Constants

The PDI library functions return the error code constants shown in the following table to indicate their status.

<b>Constant</b>	<b>Description</b>
pdiErrRead	An error occurred while reading from the input stream.
pdiErrWrite	An error occurred while writing to the output stream.
pdiErrNoPropertyName	An attempt was made to write a property value before the property name was written.
pdiErrNoPropertyValue	The application did not write the last property value.
pdiErrMoreChars	The buffer is full. Superfluous characters have been discarded.
pdiErrNoMoreFields	There are no more property fields to read.
pdiErrOpenFailed	The PDI library could not be opened.
pdiErrCloseFailed	The PDI library could not be closed. This can occur if another application is using the library.

---

## Parameter Name Constants

The PdiConst.h file defines several parameter name constants that you can use to specify the name of a parameter in functions that use parameter names. The parameter name constants have the following format:

`kPdiPAN_parameterName`

where *parameterValue* is replaced by a parameter name.

The following table shows examples of parameter name constants. For a complete list, see the PdiConst.h file.

Constant	Description
<code>kPdiPAN_TYPE</code>	The TYPE parameter.
<code>kPdiPAN_ENCODING</code>	The ENCODING parameter.
<code>kPdiPAN_STATUS</code>	The STATUS parameter.

## Parameter Value Constants

The PdiConst.h file defines several parameter value constants that you can use to specify the name and value of a parameter in functions that use name and value pairs. The parameter value constants have the following format:

`kPdiPAV_parameterName_parameterValue`

where *parameterName* is replaced by a parameter name and *parameterValue* is replaced by a parameter value.

The following table shows examples of parameter value constants. For a complete list, see the PdiConst.h file.

Constant	Description
<code>kPdiPAV_TYPE_TEL</code>	The parameter name is TYPE and the parameter value is TEL.
<code>kPdiPAV_ENCODING_BASE64</code>	The parameter name is ENCODING and parameter value is BASE64.

Constant	Description
kPdiPAV_ENCODING_8BIT	The parameter name is ENCODING and the parameter value is 8BIT.
kPdiPAV_STATUS_ACCEPTED	The parameter name is STATUS and the parameter value is ACCEPTED.

## Property Name Constants

The PdiConst.h file defines several property name constants that you can use to specify the name of a PDI property in functions that use property names. The property name constants have the following format:

kPdiPRN\_*propertyName*

where *propertyName* is replaced by a property name.

The following table shows examples of property name constants. For a complete list, see the PdiConst.h file.

Constant	Description
kPdiPRN_ADR	The ADR property.
kPdiPRN_BDAY	The BDAY property.
kPdiPRN_BEGIN	The BEGIN property.
kPdiPRN_BEGIN_VCARD	The BEGIN:VCARD property.

## Property Type Constants

You use the property type constants to specify the data type of a property.

Constant	Value	Description
kPdiType_URI	0	The data is a uniform resource identifier.
kPdiType_UTC_OFFSET	1	The data is an offset from UTC to local time.

## Personal Data Interchange Library

### PDI Library Constants

---

Constant	Value	Description
kPdiType_RECUR	2	The data is a specification of a recurrence rule.
kPdiType_PERIOD	3	The data is a precise period of time.
kPdiType_CAL_ADDRESS	4	Calendar user data.
kPdiType_BINARY	5	Binary data.
kPdiType_TEXT	6	Text data.
kPdiType_FLOAT	7	Floating-point data.
kPdiType_DURATION	8	Time duration data.
kPdiType_DATE_TIME	9	Calendar date and time data.
kPdiType_DATE	10	Date data.
kPdiType_INTEGER	11	Signed integer data.
kPdiType_TIME	12	Time-of-day data.

## Property Value Field Constants

The PdiConst.h file defines several property value field constants that you can use to specify the position of a PDI property value field in functions that use fields. The property value field constants have the following format:

`kPdiPVF_propertyValueField`

where *propertyValueField* is replaced by a property value field name.

The following table shows examples of property name constants. For a complete list, see the PdiConst.h file.

<b>Constant</b>	<b>Description</b>
kPdiPVF_ADR_POST_OFFICE	The property value field that stores the post office portion of the address.
kPdiPVF_ADR_EXTENDED	The property value field that stores the extended portion of the address.
kPdiPAN_ADR_COUNTRY	The property value field that stores the country portion of the address.

## **Property Value Format Constants**

Some properties have **structured values**, which are values that contain multiple fields. These fields are typically separated by commas or semicolons in the vObject input or output stream. You use the property value format constants with the [PdiReadPropertyField](#) and [PdiWritePropertyStr](#) functions to specify how to handle fields in a structured value.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
kPdiNoFields	0	There are no fields in the property value; <a href="#">PdiReadPropertyField</a> reads the entire value, or <a href="#">PdiWritePropertyStr</a> specifies that the entire value should be written.
kPdiCommaFields	1	Fields are separated with comma (",") characters; <a href="#">PdiReadPropertyField</a> reads one field, or <a href="#">PdiWritePropertyStr</a> specifies that one field should be written.

## Personal Data Interchange Library

### PDI Library Constants

---

Constant	Value	Description
kPdiSemicolonFields	2	Fields are separated with semicolon (“;”) characters; <a href="#">PdiReadPropertyField</a> reads one field, or <a href="#">PdiWritePropertyStr</a> specifies that one field should be written.
kPdiDefaultFields	4	The parser decides the property value format, based on the property name.
kPdiConvertComma	8	Fields are separated with comma characters; <a href="#">PdiReadPropertyField</a> reads the entire value and converts each comma into a newline (“\n”) character.
kPdiConvertSemicolon	16	Fields are separated with semicolon characters; <a href="#">PdiReadPropertyField</a> reads the entire value and converts each semicolon into a newline (“\n”) character.

---

## Reader and Writer Options Constants

You use the reader and writer option constants to control how the PDI reader (parser) reads values from the input stream or to control how the PDI writer (generator) writes values to the output stream.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
kPdiEnableFolding	1	<p>Enables folding of properties in the output stream.</p> <p><b>Folding</b> is a mechanism for breaking long lines to allow them to be transmitted without change. If you specify this flag, the PDI library folds long lines.</p> <p>Note that folding is not compatible with earlier versions of the Palm OS®.</p> <p>Also note that other encoding formats, including quoted-printable and Base 64, define their own mechanisms for splitting long lines.</p>
kPdiEnableQuotedPrintable	2	<p>Enables quoted-printable encoding in the output stream and makes it the default encoding.</p> <p>This is an encoding format for non-ASCII values. You must have this enabled for compatibility with earlier versions of the Palm OS.</p> <p>If you do not specify this option, the default encoding is Base 64.</p>

## Personal Data Interchange Library

### PDI Library Constants

---

Constant	Value	Description
kPdiEscapeMultiFieldValues	4	For compatibility with earlier versions of the Palm OS.  You must enable this for compatibility with earlier versions of the Palm OS. However, some non-Palm PDI software does not support this format.
		For more information about compatibility with earlier versions of the Palm OS, see <a href="#">Format Compatibility</a> in the <i>Palm OS Programmer's Companion</i> , vol. II, <i>Communications</i> .
kPdiPalmCompatibility	6	This is a combination of kPdiEnableQuotedPrintable   kPdiEscapeMultiFieldValues.  It forces the PDI writer to generate output that is compatible with earlier versions of the Palm OS.
kPdiEnableB	8	Enables base 64 encoding in the output stream, and tells the PDI writer to output "ENCODING=B" instead of "ENCODING=BASE64" when encoding a value with Base 64.  Note: the vCard 3.0 standard has replaced the earlier ENCODING=BASE64 with ENCODING=B. The meaning is the same.
kPdiOpenParser	8	Specifies that the PDI reader is open to all formats, including Palm and others.

---

## Reader Event Constants

The PDI reader event constants specify the events that the reader has handled during the current read operation. The event values are combined together and stored in the `events` field of the PDI reader object. You can use them to test whether the reader handled a certain event.

<b>Constant</b>	<b>Value</b>	<b>Description</b>
<code>kPdiEOFEventMask</code>	1	End of file was reached.
<code>kPdiGroupNameEventMask</code>	2	A group name was found.
<code>kPdiPropertyNameEventMask</code>	4	A property name was found.
<code>kPdiParameterNameEventMask</code>	8	A parameter name was found.
<code>kPdiParameterValueEventMask</code>	16	A parameter value was found.
<code>kPdiPropertyDefinedEventMask</code>	32	A property definition was found; this implies that the ":" separator character was found.
<code>kPdiPropertyValueEventMask</code>	64	An entire property value was found
<code>kPdiPropertyValueFieldEventMask</code>	128	A value field was found; this implies that the ";" separator character was found.
<code>kPdiPropertyValueItemEventMask</code>	256	A value item was found; this implies that the "," separator character was found.

## Personal Data Interchange Library

### PDI Library Constants

---

Constant	Value	Description
kPdiPropertyValueMoreCharsEventMask	512	The buffer that you provided was not large enough. The superfluous characters have been discarded.
kPdiBeginObjectEventMask	1024	The object begin indicator BEGIN was reached.
kPdiEndObjectEventMask	2048	The object end indicator END was reached.

---

## Value Type Constants

You can use the following constants to specify data typing information for the [PdiWritePropertyBinaryValue](#), [PdiWritePropertyFields](#), and [PdiWritePropertyValue](#) functions.

Constant	Value	Description
kPdiWriteData	0	The value is data. The PDI writer does not compute a character set. You can use this for binary data or pure ASCII data.
kPdiWriteText	8	The value is text data. The PDI writer parses the data character by character to compute the correct charset and character encoding for the data.
kPdiWriteMultiline	16	Explicitly specifies that the value contains special characters, such as newlines, and must be encoded. If this flag is not specified, the encoding is determined by the applied character set.

---

# PDI Library Functions

## PdiDefineReaderDictionary

<b>Purpose</b>	Installs a new custom dictionary for use with a PDI reader object.								
<b>Declared In</b>	PdiLib.h								
<b>Prototype</b>	<pre>PdiDictionary *PdiDefineReaderDictionary (UInt16 libRefnum, PdiReaderType *ioReader, PdiDictionary *dictionary, Boolean disableMainDictionary)</pre>								
<b>Parameters</b>	<table><tr><td>-&gt; libRefnum</td><td>The PDI library reference number.</td></tr><tr><td>-&gt; ioReader</td><td>The PDI reader object with which to associate the dictionary. This object must have previously been created by a call to the <a href="#">PdiReaderNew</a> function.</td></tr><tr><td>-&gt; dictionary</td><td>A pointer to a dictionary object that was created by the . The dictionary object is an array of binary data.</td></tr><tr><td>-&gt; disableMainDictionary</td><td>If true, the main reader dictionary is disabled, and only this new dictionary is searched for terms; if false, the new dictionary supplements the main dictionary.</td></tr></table>	-> libRefnum	The PDI library reference number.	-> ioReader	The PDI reader object with which to associate the dictionary. This object must have previously been created by a call to the <a href="#">PdiReaderNew</a> function.	-> dictionary	A pointer to a dictionary object that was created by the . The dictionary object is an array of binary data.	-> disableMainDictionary	If true, the main reader dictionary is disabled, and only this new dictionary is searched for terms; if false, the new dictionary supplements the main dictionary.
-> libRefnum	The PDI library reference number.								
-> ioReader	The PDI reader object with which to associate the dictionary. This object must have previously been created by a call to the <a href="#">PdiReaderNew</a> function.								
-> dictionary	A pointer to a dictionary object that was created by the . The dictionary object is an array of binary data.								
-> disableMainDictionary	If true, the main reader dictionary is disabled, and only this new dictionary is searched for terms; if false, the new dictionary supplements the main dictionary.								
<b>Result</b>	Returns a pointer to the previously installed custom dictionary, or NULL if there was not a previously installed custom dictionary.								
<b>Comments</b>	<p>This function installs a dictionary for use with the <code>ioReader</code> object. The dictionary contains the syntax for extensions or replacements of the PDI properties about which the PDI reader knows. The reader knows about properties specified in one of the vObject standards, including the vCard or vCal standards.</p> <p>You can uninstall the current custom dictionary by specifying NULL as the value of the <code>dictionary</code> parameter,</p>								

## Personal Data Interchange Library

### PDI Library Functions

---

Note that the dictionary must have previously been compiled by the dictionary tool. For more information, review the PDI sample code, which you can find on the Palm Developer's Knowledge Base at <http://www.palmos.com/dev/tech/kb>.

## PdiDefineResizing

**Purpose** Defines the sizing information to use when automatically resizing a buffer. PDI reader objects read information from the input stream into a buffer and automatically adjust the buffer size as required.

**Declared In** PdiLib.h

**Prototype** Err PdiDefineResizing(UInt16 libRefnum,  
PdiReaderType \*ioReader, UInt16 deltaSize,  
UInt16 maxSize)

**Parameters**

-> libRefnum	The PDI library reference number.
-> ioReader	The PDI reader object, which was created by a previous call to the <a href="#">PdiReaderNew</a> function.
-> deltaSize	The number of bytes by which to grow the buffer when it needs resizing.
-> maxSize	The maximum allowable size for the buffer.

**Result** Returns errNone if successful, and an error code if not successful.

**Comments** This function redefines the values to use when resizing a buffer. It does not perform any other actions.

The resizing values are used if your reader runs out of buffer space when storing input data during the processing of a property value. If possible, the reader resizes its internal buffer, using the values that you supply in this function.

The default resizing values apply if you do not call this function. The default values are:

kPdiDefaultBufferDeltaSize	0x0010
kPdiDefaultBufferMaxSize	0x3FFF

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## **PdiDefineWriterDictionary**

**Purpose** Installs a new custom dictionary for use with a PDI writer object.

**Declared In** PdiLib.h

**Prototype** `PdiDictionary *PdiDefineWriterDictionary  
(UInt16 libRefnum, PdiWriterType *ioWriter,  
PdiDictionary *dictionary,  
Boolean disableMainDictionary)`

**Parameters**

-> libRefnum	The PDI library reference number.
-> ioWriter	The PDI writer object with which to associate the dictionary. This object must have previously been created by a call to the <a href="#">PdiWriterNew</a> function.
-> dictionary	A pointer to a dictionary object that was created by the . The dictionary object is an array of binary data.
-> disableMainDictionary	If true, the main dictionary is disabled, and only this new dictionary is searched for terms; if false, the new dictionary supplements the main dictionary.

**Result** Returns a pointer to the previously installed custom dictionary, or NULL if there was not a previously installed custom dictionary.

**Comments** This function installs a dictionary for use with the `ioWriter` object. The dictionary contains the syntax for extensions or replacements of the PDI properties about which the PDI writer knows. The writer knows about properties specified in one of the vObject standards, including the vCard or vCal standards.

You can uninstall the current custom dictionary by specifying NULL as the value of the `dictionary` parameter,

## Personal Data Interchange Library

### PDI Library Functions

---

Note that the dictionary must have previously been compiled by the dictionary tool. For more information, review the PDI sample code, which you can find on the Palm Developer's Knowledge Base at <http://www.palmos.com/dev/tech/kb/>.

## PdiEnterObject

**Purpose** Tells the PDI library to enter into a recursively-defined object.

**Declared In** PdiLib.h

**Prototype** Err PdiEnterObject (UInt16 libRefnum,  
PdiReaderType \*ioReader)

**Parameters** -> libRefnum The PDI library reference number.  
-> ioReader The PDI reader object, which was created by a previous call to the [PdiReaderNew](#) function.

**Result** Returns errNone if successful, and an error code if not successful.

**Comments** Some vObjects recursively define other vObjects. Your application can choose whether or not to enter and parse the recursively defined objects.

If you want to parse the nested object definition, you need to call this function; otherwise, all of the properties of the nested object are skipped when the next call is made to the [PdiReadProperty](#) or [PdiReadPropertyName](#) functions.

Call this function after a BEGIN\_VObject statement of the nested object has been parsed.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## PdiLibClose

<b>Purpose</b>	Close the PDI library after your application has finished with it.
<b>Declared In</b>	PdiLib.h
<b>Prototype</b>	Err PdiLibClose (UInt16 libRefnum)
<b>Parameters</b>	-> libRefnum     The PDI library reference number.
<b>Result</b>	Returns 0 if no other application uses the library. You may need to call <a href="#">SysLibRemove</a> to remove the PDI library from the system library table.  Returns pdiErrCloseFailed if the library could not be closed.
<b>Comments</b>	You should call this function after your application has finished with the PDI library, to allow the resources to be recovered.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">PdiLibOpen</a>

## PdiLibOpen

<b>Purpose</b>	Opens the PDI library for use by your application.
<b>Declared In</b>	PdiLib.h
<b>Prototype</b>	Err PdiLibOpen (UInt16 libRefnum)
<b>Parameters</b>	-> libRefnum     The PDI library reference number.
<b>Result</b>	Returns errNone if successful, and an error code if not successful.
<b>Comments</b>	You must call this function before calling any of the other PDI functions.

## Personal Data Interchange Library

### PDI Library Functions

---

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [PdiLibClose](#)

## PdiParameterPairTest

**Purpose** A macro that determines if the reader has already parsed the specified parameter value or name-value pair.

**Declared In** PdiLib.h

**Prototype** PdiParameterPairTest (reader, pair)

**Parameters**

-> reader	The PDI reader object, which was created by a previous call to the <a href="#">PdiReaderNew</a> function.
-> pair	The ID of the parameter. This must be one of the <a href="#">Parameter Value Constants</a> .

**Result** Returns `true` if the specified parameter name-value pair has been parsed for the current property, and `false` if not.

**Comments** Some vObject generators do not specify the parameter name if the name is considered evident from the context. This means that both of the following constructs are considered proper:

Name=Value  
Value

The PdiParameterPairTest macro returns `true` if the value has been parsed in either format. For example,

```
PdiParameterPairTest(reader, kPdiPAV_TYPE_WORK)  
returns true for either of the following:
```

Type=WORK  
WORK

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## PdiReaderDelete

<b>Purpose</b>	Delete a PDI reader object that is associated with the specified library number.				
<b>Declared In</b>	PdiLib.h				
<b>Prototype</b>	<code>void PdiReaderDelete(UInt16 libRefnum, PdiReaderType **ioReader)</code>				
<b>Parameters</b>	<table><tr><td>-&gt; libRefnum</td><td>The PDI library reference number.</td></tr><tr><td>&lt;-&gt; ioReader</td><td>A pointer to the PDI reader object, which was created by a previous call to the <a href="#">PdiReaderNew</a> function.</td></tr></table>	-> libRefnum	The PDI library reference number.	<-> ioReader	A pointer to the PDI reader object, which was created by a previous call to the <a href="#">PdiReaderNew</a> function.
-> libRefnum	The PDI library reference number.				
<-> ioReader	A pointer to the PDI reader object, which was created by a previous call to the <a href="#">PdiReaderNew</a> function.				
<b>Result</b>	Returns nothing.				
<b>Comments</b>	This function deletes the UDAReader object associated with the reader object and frees the memory that was allocated for the reader object. The <code>ioReader</code> parameter is set to NULL.				
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.				
<b>See Also</b>	<a href="#">PdiReaderNew</a>				

## PdiReaderNew

<b>Purpose</b>	Create and initialize a new PDI reader object for use with the specified PDI library number.		
<b>Declared In</b>	PdiLib.h		
<b>Prototype</b>	<code>PdiReaderType *PdiReaderNew(UInt16 libRefnum, UDAReaderType *input, UInt16 version)</code>		
<b>Parameters</b>	<table><tr><td>-&gt; libRefnum</td><td>The PDI library reference number.</td></tr></table>	-> libRefnum	The PDI library reference number.
-> libRefnum	The PDI library reference number.		

## Personal Data Interchange Library

### PDI Library Functions

---

-> input	The Unified Data Access (UDA) input object associated with the reader.
-> version	Options to control the parsing behavior of the reader. You can use a combination of the <a href="#">Reader and Writer Options Constants</a> .

**Result** Returns a pointer to the new PDI reader object. Returns NULL if the reader cannot be created.

**Comments** The current implementation of the PdiReaderNew function does not make use of the optionFlags settings because the reader knows how to adapt itself to all of the supported formats. The options will be used in future versions.

The input value is a UDA object for reading data from an input stream that can be connected to various data sources. For example, you can use a UDAExchangeReader to read data from the Exchange Manager, and you can use a UDAStringReader to read data from a string. For more information about the UDA Manager, see [Chapter 86, “Unified Data Access Manager.”](#)

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [PdiReaderDelete](#), [PdiWriterNew](#)

## PdiReadParameter

**Purpose** Read a single parameter name and its value from an input stream.

**Declared In** PdiLib.h

**Prototype** Err PdiReadParameter(UINT16 libRefnum,  
PdiReaderType \*ioReader)

**Parameters** -> libRefnum The PDI library reference number.

	- > ioReader	The PDI reader object, which was created by a previous call to the <a href="#">PdiReaderNew</a> function.
<b>Result</b>	0	The parameter and its value were read successfully.
	kPdiReadError	The parameter and its value could not be successfully read from the input stream.
		This function initializes the <code>parameterName</code> and <code>parameter</code> fields of the <code>ioReader</code> object.
		This function sets the appropriate bits in the reader's <code>parameterValues</code> field if the parameter name is recognized.
<b>Compatibility</b>		Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>		<a href="#">PdiReaderNew</a> , <a href="#">PdiReadPropertyField</a> , <a href="#">PdiReadPropertyName</a>

## PdiReadProperty

<b>Purpose</b>	Read the next property and its parameters from the input stream.
<b>Declared In</b>	PdiLib.h
<b>Prototype</b>	Err PdiReadProperty(UInt16 libRefnum, PdiReaderType *ioReader)
<b>Parameters</b>	- > libRefnum      The PDI library reference number. - > ioReader      The PDI reader object, which was created by a previous call to the <a href="#">PdiReaderNew</a> function.
<b>Result</b>	Returns <code>errNone</code> if successful. Returns <code>kPdiReadError</code> if an error occurs.
<b>Comments</b>	The <code>PdiReadProperty</code> function reads a property name and its parameters, by reading until it encounters the PDI ":" separator character.

## Personal Data Interchange Library

### PDI Library Functions

---

This function looks each name up in the properties dictionary, and sets the appropriate bit in the `ioReader` object structure to indicate that property-parameter pair has been read. The properties dictionary stores information about properties that are considered well known, as described in [The PDI Library Properties Dictionary](#) in [Chapter 3, “Personal Data Interchange,”](#) in *Palm OS Programmer’s Companion*, vol. II, *Communications*.

To read a property, you call `PdiReadProperty`, followed by a call or calls to the [PdiReadPropertyField](#) function to read the property value. For more information, see [Reading Properties](#) in [Chapter 3, “Personal Data Interchange,”](#) in *Palm OS Programmer’s Companion*, vol. II, *Communications*.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [PdiReaderNew](#), [PdiReadPropertyName](#), [PdiReadParameter](#)

## PdiReadPropertyField

**Purpose** Read one field of a property value. The property value can be structured to contain multiple fields that are separated by commas or semicolons.

**Declared In** `PdiLib.h`

**Prototype** `Err PdiReadPropertyField(UInt16 libRefnum,  
PdiReaderType *ioReader, Char **bufferPP,  
UInt16 bufferSize, UInt16 readMode)`

**Parameters**

<code>-&gt; libRefnum</code>	The PDI library reference number.
<code>-&gt; ioReader</code>	The PDI reader object, which was created by a previous call to the <a href="#">PdiReaderNew</a> function.
<code>&lt;-&gt; bufferPP</code>	A pointer to a pointer to the buffer into which the field characters are stored. Set this value to <code>NULL</code> to allow the PDI library to manage it.

Note that the PDI library may need to resize the buffer; thus, the value of this parameter might change.

-> bufferSize The size, in bytes, of the input buffer for reading the field.

You can use the `PdiResizableBuffer` constant to specify that the PDI Library can automatically resize the buffer as required.

If you do not specify the `PdiResizableBuffer` value, then the PDI library assumes that buffer cannot be moved, and that its size is fixed.

-> readMode The format of the fields in the property value. Use one of the [Property Value Format Constants](#).

**Result** 0 The field was read successfully.

`kPdiNoMoreFieldsError`

There are no more fields to read because the entire value has already been read.

`kPdiMoreCharsError`

The buffer is not large enough to store the entire field.

**Comments** The value returned in the buffer is terminated with the “\0” character.

If the field is an empty string, the buffer is erased from memory, and the value of buffer is set to NULL.

If you specify `kPdiResizableBuffer` for the value of the `bufferSize` parameter, and the buffer needs more space, `PdiReadPropertyField` resizes the buffer for you, which may cause the value of buffer to be modified.

This function initializes the `propertyName` and `fieldNum` fields of the `ioReader` object.

To read a property, you usually call the [PdiReadProperty](#) function, followed by a call or calls to `PdiReadPropertyField` to

## Personal Data Interchange Library

### PDI Library Functions

---

read the property value. For more information, see [Reading Properties in Chapter 3, "Personal Data Interchange,"](#) in *Palm OS Programmer's Companion*, vol. II, *Communications*.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [PdiReaderNew](#), [PdiReadProperty](#), [PdiReadPropertyName](#), [PdiReadParameter](#)

## PdiReadPropertyName

**Purpose** Read a property name from an input stream. Use this function when you want to parse and process each parameter individually.

**Declared In** PdiLib.h

**Prototype** Err PdiReadPropertyName (UInt16 libRefnum,  
PdiReaderType \*ioReader)

**Parameters** -> libRefnum The PDI library reference number.  
-> ioReader The PDI reader object, which was created by a previous call to the [PdiReaderNew](#) function.

**Result** Returns errNone if successful, and an error code if not successful.

**Comments** The PdiReadProperty function reads a property name only, reading until it encounters the PDI ":" separator character, or until it encounters the first parameter "," separator character.  
This function initializes the property and propertyName fields of the ioReader object.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [PdiReaderNew](#), [PdiReadPropertyField](#), [PdiReadProperty](#), [PdiReadParameter](#)

## PdiSetCharset

<b>Purpose</b>	Force the character set of the next property value that is written by the specified PDI writer.						
<b>Declared In</b>	PdiLib.h						
<b>Prototype</b>	<pre>Err PdiSetCharset (UInt16 libRefnum, PdiWriterType *ioWriter, CharEncodingType charset)</pre>						
<b>Parameters</b>	<table><tr><td>-&gt; libRefnum</td><td>The PDI library reference number.</td></tr><tr><td>-&gt; ioWriter</td><td>The PDI writer object, which was created by a previous call to the <a href="#">PdiWriterNew</a> function.</td></tr><tr><td>-&gt; charset</td><td>The character set to use for the property value. This must be one of the following <a href="#">CharEncodingType</a> values: charEncodingAscii charEncodingISO8859_1 charEncodingShiftJIS charEncodingISO2022Jp</td></tr></table>	-> libRefnum	The PDI library reference number.	-> ioWriter	The PDI writer object, which was created by a previous call to the <a href="#">PdiWriterNew</a> function.	-> charset	The character set to use for the property value. This must be one of the following <a href="#">CharEncodingType</a> values: charEncodingAscii charEncodingISO8859_1 charEncodingShiftJIS charEncodingISO2022Jp
-> libRefnum	The PDI library reference number.						
-> ioWriter	The PDI writer object, which was created by a previous call to the <a href="#">PdiWriterNew</a> function.						
-> charset	The character set to use for the property value. This must be one of the following <a href="#">CharEncodingType</a> values: charEncodingAscii charEncodingISO8859_1 charEncodingShiftJIS charEncodingISO2022Jp						
<b>Result</b>	Returns errNone if successful, and an error code if not successful.						
<b>Comments</b>	This function tells ioWriter to use the specified charSet for the next property value that it writes, rather than computing a character set for that value.  You can determine the current character setting by examining the charset field of your PDI writer object.						
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.						
<b>See Also</b>	<a href="#">PdiSetEncoding</a>						

## PdiSetEncoding

**Purpose** Force the encoding of the current property value.

**Declared In** PdiLib.h

**Prototype** Err PdiSetEncoding(UInt16 libRefnum,  
PdiWriterType \*ioWriter, UInt16 encoding)

**Parameters**

-> libRefnum	The PDI library reference number.
-> ioReader	The PDI writer object, which was created by a previous call to the <a href="#">PdiWriterNew</a> function.
-> encoding	The encoding to apply to the property value. This must be one of the <a href="#">Encoding Type Constants</a> .

**Result** Returns errNone if successful, and an error code if not successful.

**Comments** This function changes the encoding for the property value to the specified encoding value

You can determine the current encoding setting by examining the encoding field of your PDI writer object.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [PdiSetCharset](#)

## PdiWriteBeginObject

**Purpose** Writes a vObject begin tag to an output stream.

**Declared In** PdiLib.h

**Prototype** Err PdiWriteBeginObject(UInt16 libRefnum,  
PdiWriterType \*ioWriter, UInt16 objectNameID)

**Parameters**

-> libRefnum	The PDI library reference number.
--------------	-----------------------------------

-> `ioWriter` The PDI writer object, which was created by a previous call to the [PdiWriterNew](#) function.

-> `objectNameID` The object name ID. This must be one of the [Property Name Constants](#) that begins an object, including the following:

`kPdiPRN_BEGIN_VCAL`  
`kPdiPRN_BEGIN_VCARD`  
`kPdiPRN_BEGIN_VEVENT`  
`kPdiPRN_BEGIN_VFREEBUSY`  
`kPdiPRN_BEGIN_VJOURNAL`  
`kPdiPRN_BEGIN_VTIMEZONE`  
`kPdiPRN_BEGIN_VTODO`

**Result** Returns `errNone` if successful, and an error code if not successful.

**Comments** Call this function to begin writing a vObject to the output stream. It writes a begin tag such as “`BEGIN:VCARD`” to the output stream.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [PdiWriteEndObject](#), [PdiWriteProperty](#)

## **PdiWriteEndObject**

**Purpose** Writes a vObject end tag to an output stream.

**Declared In** `PdiLib.h`

**Prototype** `Err PdiWriteEndObject (UInt16 libRefnum,  
PdiWriterType *ioWriter, UInt16 objectNameID)`

**Parameters** -> `libRefnum` The PDI library reference number.

## Personal Data Interchange Library

### PDI Library Functions

---

-> ioWriter      The PDI writer object, which was created by a previous call to the [PdiWriterNew](#) function.

-> objectNameID      The object name ID. This must be one of the [Property Name Constants](#) that ends an object, including the following:

kPdiPRN\_END\_VCAL  
kPdiPRN\_END\_VCARD  
kPdiPRN\_END\_VEVENT  
kPdiPRN\_END\_VFREEBUSY  
kPdiPRN\_END\_VJOURNAL  
kPdiPRN\_END\_VTIMEZONE  
kPdiPRN\_END\_VTODO

**Result**      Returns errNone if successful, and an error code if not successful.

**Comments**      Call this function to finish writing a vObject to the output stream. It writes a end tag such as "END : VCARD" to the output stream.

**Compatibility**      Implemented only if [4.0 New Feature Set](#) is present.

**See Also**      [PdiWriteBeginObject](#), [PdiWriteProperty](#)

## PdiWriteParameter

**Purpose**      Write a parameter, and optionally its name, to an output stream.

**Declared In**      PdiLib.h

**Prototype**      Err PdiWriteParameter(UInt16 libRefnum,  
                  PdiWriterType \*ioWriter, UInt16 parameter,  
                  Boolean parameterName)

**Parameters**      -> libRefnum      The PDI library reference number.

-> ioWriter	The PDI writer object, which was created by a previous call to the <a href="#">PdiWriterNew</a> function.
-> parameter	The ID of the parameter. This must be one of the <a href="#">Parameter Value Constants</a> .
-> parameterName	If this is <code>true</code> , the parameter name, followed by the “=” symbol, followed by the parameter value is written to the output stream. If this is <code>false</code> , only the parameter value is written to the output stream.

**Result** Returns `errNone` if successful, and an error code if not successful.

**Comments** Use this function to write a parameter to the output stream. To write a property, you usually call the [PdiWriteProperty](#) function, followed by calls to [PdiWriteParameter](#) to write any parameters, followed by a call to the [PdiWritePropertyValue](#) function to write the property value. For more information, see [Writing Properties in Chapter 3, “Personal Data Interchange,” in Palm OS Programmer’s Companion](#), vol. II, *Communications*.

You can use the `parameterName` argument to specify that you want the parameter name written as well as the parameter value. For example, the following table shows what is written if the value of `parameter` is `kPdiPAV_TYPE_HOME`.

<b>Value of parameterName</b>	<b>Data written to output stream</b>
<code>true</code>	<code>TYPE=HOME</code>
<code>false</code>	<code>HOME</code>

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [PdiWriteProperty](#), [PdiWritePropertyValue](#), [PdiWritePropertyFields](#), [PdiWritePropertyStr](#), [PdiWriteParameterStr](#)

## PdiWriteParameterStr

**Purpose** Write a parameter name and the parameter value to an output stream.

**Declared In** PdiLib.h

**Prototype** Err PdiWriteParameterStr(UInt16 libRefnum,  
PdiWriterType \*ioWriter,  
const Char \*parameterName,  
const Char \*parameterValue)

**Parameters**

- > libRefnum The PDI library reference number.
- > ioWriter The PDI writer object, which was created by a previous call to the [PdiWriterNew](#) function.
- > parameterName The name of the parameter. If the value of this is the empty string or NULL, only the parameter value is written.
- > parameterValue The parameter value string.

**Result** Returns errNone if successful, and an error code if not successful.

**Comments** This function writes the parameter name, followed by the “=” symbol, followed by the parameter value, to the output stream. If parameterName is NULL, or if its value is the empty string, just the parameter value is written.  
This function is similar to the [PdiWriteParameter](#) function. The difference is that PdiWriteParameterStr takes the name and value of the parameter as strings, while PdiWriteParameter takes them as ID constants.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [PdiWriteProperty](#), [PdiWritePropertyValue](#),  
[PdiWritePropertyFields](#), [PdiWritePropertyStr](#),  
[PdiWriteParameter](#)

## PdiWriteProperty

**Purpose** Writes a property name to an output stream.

**Declared In** PdiLib.h

**Prototype** Err PdiWriteProperty(UInt16 libRefnum,  
PdiWriterType \*ioWriter, UInt16 propertyNameID)

**Parameters**

-> libRefnum	The PDI library reference number.
-> ioWriter	The PDI writer object, which was created by a previous call to the <a href="#">PdiWriterNew</a> function.
-> propertyNameID	The property name to write. This must be one of the <a href="#">Property Name Constants</a> .

**Result** Returns errNone if successful, and an error code if not successful.

**Comments** To write a property, you usually call `PdiWriteProperty`, followed by calls to the [PdiWriteParameter](#) function to write any parameters, followed by a call to the [PdiWritePropertyValue](#) function to write the property value. For more information, see [Writing Properties in Chapter 3, “Personal Data Interchange,” in Palm OS Programmer’s Companion, vol. II, Communications](#).

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [PdiWritePropertyValue](#), [PdiWritePropertyFields](#),  
[PdiWritePropertyStr](#), [PdiWriteParameter](#)

## PdiWritePropertyBinaryValue

**Purpose** Write a binary property value to an output stream.

**Declared In** PdiLib.h

**Prototype** Err PdiWritePropertyBinaryValue(UInt16 libRefnum,  
PdiWriterType \*ioWriter, const Char \*buffer,  
UInt16 size, UInt16 options)

**Parameters**

-> libRefnum	The PDI library reference number.
-> ioWriter	The PDI writer object, which was created by a previous call to the <a href="#">PdiWriterNew</a> function.
-> buffer	A buffer that contains the binary data.
-> size	The number of bytes of data to write from the buffer.
-> options	The data type. This must be a combination of one or more of the <a href="#">Value Type Constants</a> .

**Result** Returns errNone if successful, and an error code if not successful.

**Comments** Use this function to write a binary data property value, such as a sound or an image.

This function encodes the data when it is written. The character set that gets applied to the data is not computed by this function; however, you can call the [PdiSetCharset](#) function to set the character set.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [PdiWriteProperty](#), [PdiWritePropertyFields](#),  
[PdiWritePropertyValue](#)

## PdiWritePropertyFields

<b>Purpose</b>	Write a structured property value with multiple fields to an output stream.										
<b>Declared In</b>	PdiLib.h										
<b>Prototype</b>	<pre>Err PdiWritePropertyFields(UInt16 libRefnum,                            PdiWriterType *ioWriter, Char *fields[],                            UInt16 fieldNumber, UInt16 options)</pre>										
<b>Parameters</b>	<table><tr><td>-&gt; libRefnum</td><td>The PDI library reference number.</td></tr><tr><td>-&gt; ioWriter</td><td>The PDI writer object, which was created by a previous call to the <a href="#">PdiWriterNew</a> function.</td></tr><tr><td>-&gt; fields</td><td>An array of strings, each of which is a field of the property value. Individual fields can be NULL.</td></tr><tr><td>-&gt; fieldNumber</td><td>The number of fields in the Fields array.</td></tr><tr><td>-&gt; options</td><td>The data type. This must be a combination of one or more of the <a href="#">Value Type Constants</a>.</td></tr></table>	-> libRefnum	The PDI library reference number.	-> ioWriter	The PDI writer object, which was created by a previous call to the <a href="#">PdiWriterNew</a> function.	-> fields	An array of strings, each of which is a field of the property value. Individual fields can be NULL.	-> fieldNumber	The number of fields in the Fields array.	-> options	The data type. This must be a combination of one or more of the <a href="#">Value Type Constants</a> .
-> libRefnum	The PDI library reference number.										
-> ioWriter	The PDI writer object, which was created by a previous call to the <a href="#">PdiWriterNew</a> function.										
-> fields	An array of strings, each of which is a field of the property value. Individual fields can be NULL.										
-> fieldNumber	The number of fields in the Fields array.										
-> options	The data type. This must be a combination of one or more of the <a href="#">Value Type Constants</a> .										
<b>Result</b>	Returns errNone if successful, and an error code if not successful.										
<b>Comments</b>	Use this function to write a property value that contains multiple fields.										
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.										
<b>See Also</b>	<a href="#">PdiWritePropertyValue</a> , <a href="#">PdiWritePropertyBinaryValue</a> , <a href="#">PdiReadPropertyField</a>										

## PdiWritePropertyStr

**Purpose** Writes the name of a property to the output stream, and specifies the property value's structure for subsequent write operations.

**Declared In** PdiLib.h

**Prototype** Err PdiWritePropertyStr(UInt16 libRefnum,  
PdiWriterType \*ioWriter,  
const Char \*propertyName, UInt8 writeMode,  
UInt8 requiredFields)

**Parameters**

- > libRefnum The PDI library reference number.
- > ioWriter The PDI writer object, which was created by a previous call to the [PdiWriterNew](#) function.
- > propertyName The name of the property to write.
- > writeMode The format of the fields in the property value. Use one of the following [Property Value Format Constants](#):
  - kPdiNoFields
  - kPdiCommaFields
  - kPdiSemicolonFields
- > requiredFields The number of required fields for the value. This is usually set to 1.

**Result** Returns errNone if successful, and an error code if not successful.

**Comments** Use this function when you are writing a property that is not in the dictionary, or when you are writing a property that uses value formatting that differs from the standard formatting stored in the dictionary for the property name.

This function writes the property name to the output stream, and then establishes the structure of the property's value, including the number of required fields and the separator character for those

fields. After calling this function, the next call to the [PdiWritePropertyValue](#) or [PdiWritePropertyFields](#) functions correctly writes the property value.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [PdiWriteProperty](#), [PdiWritePropertyValue](#),  
[PdiWritePropertyFields](#), [PdiWriteParameter](#)

## PdiWritePropertyValue

**Purpose** Write a string to the output stream as the entire value of a property.

**Declared In** PdiLib.h

**Prototype** Err PdiWritePropertyValue(UINT16 libRefnum,  
PdiWriterType \*ioWriter, Char \*buffer,  
UINT16 options)

**Parameters**

-> libRefnum	The PDI library reference number.
-> ioWriter	The PDI writer object, which was created by a previous call to the <a href="#">PdiWriterNew</a> function.
-> buffer	The input buffer that contains the string to be written.
-> options	The data type. This must be a combination of one or more of the <a href="#">Value Type Constants</a> .

**Result** Returns errNone if successful, and an error code if not successful.

**Comments** Use this function to write a property value that contains a single field.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [PdiWriteProperty](#), [PdiWritePropertyFields](#),  
[PdiWriteParameter](#), [PdiWritePropertyValueBinaryValue](#)

## Personal Data Interchange Library

### PDI Library Functions

---

## PdiWriterDelete

<b>Purpose</b>	Delete a PDI output stream object.				
<b>Declared In</b>	PdiLib.h				
<b>Prototype</b>	<code>void PdiWriterDelete(UINT16 libRefnum, PdiWriterType **ioWriter)</code>				
<b>Parameters</b>	<table><tr><td>-&gt; libRefnum</td><td>The PDI library reference number.</td></tr><tr><td>&lt;-&gt; ioWriter</td><td>A pointer to the PDI writer object, which was created by a previous call to the <a href="#">PdiWriterNew</a> function.</td></tr></table>	-> libRefnum	The PDI library reference number.	<-> ioWriter	A pointer to the PDI writer object, which was created by a previous call to the <a href="#">PdiWriterNew</a> function.
-> libRefnum	The PDI library reference number.				
<-> ioWriter	A pointer to the PDI writer object, which was created by a previous call to the <a href="#">PdiWriterNew</a> function.				
<b>Result</b>	Returns nothing.				
<b>Comments</b>	This function frees the memory that was allocated for the writer object. The <code>ioWriter</code> parameter is set to NULL.				
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.				
<b>See Also</b>	<a href="#">PdiWriterNew</a>				

## PdiWriterNew

<b>Purpose</b>	Initializes a new PDI writer object for use with the specified library number.				
<b>Declared In</b>	PdiLib.h				
<b>Prototype</b>	<code>PdiWriterType *PdiWriterNew(UINT16 libRefnum, UDAWriterType *output, UINT16 version)</code>				
<b>Parameters</b>	<table><tr><td>-&gt; libRefnum</td><td>The PDI library reference number.</td></tr><tr><td>-&gt; output</td><td>The Unified Data Access (UDA) output object associated with the writer.</td></tr></table>	-> libRefnum	The PDI library reference number.	-> output	The Unified Data Access (UDA) output object associated with the writer.
-> libRefnum	The PDI library reference number.				
-> output	The Unified Data Access (UDA) output object associated with the writer.				

-> version

Options to control the behavior of the writer.  
You can use a combination of the [Reader and Writer Options Constants](#).

**Result** Returns a pointer to the new PDI writer object. Returns NULL if the reader cannot be created.

**Comments** The media pointer is copied into a field in the writer object; thus, you do not need to retain your copy.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [PdiWriterDelete](#), [PdiReaderNew](#)

## **Personal Data Interchange Library**

### *PDI Library Functions*

---

# Unified Data Access Manager

---

This chapter provides reference material for the Unified Data Access (UDA) Manager, which provides a mechanism for abstracting read and write access to different kinds of source and destination media, including memory and the Exchange Manager.

The Personal Data Interchange (PDI) reader and writer objects use UDA objects, and you must create UDA objects to use the PDI functions.

This chapter discusses the following topics:

- [UDA Manager Data Structures](#)
- [UDA Manager Constants](#)
- [UDA Manager Functions](#)
- [UDA Object Creation Functions](#)

The header file `UDAMgr.h` declares the Unified Data Access Manager API.

You use the UDA Manager in conjunction with the PDI library. For more information about the PDI library, see [Chapter 85, “Personal Data Interchange Library.”](#)

[Chapter 3, “Personal Data Interchange,”](#) in the *Palm OS Programmer’s Companion*, vol. II, *Communications*, provides examples of using the UDA functions with the PDI library.

## UDA Manager Data Structures

### **UDABufferSize**

The `UDABufferSize` type is a simple typedef that defines the size of buffers used with UDA read functions.

## Unified Data Access Manager

### UDA Manager Data Structures

---

```
typedef UInt16 UDABufferSize
```

## UDAObjectType

The UDAObjectType is the base class for all UDA objects, and defines the common properties of all of the objects.

```
typedef struct UDAObjectType {
    UInt16          optionFlags;
    UDADeleteFunction deleteF;
    UDAControlFunction controlF;
} UDAObjectType
```

### Field Descriptions

optionFlags	Options for the object. This is a combination of values described in <a href="#">Object Option Flags</a> .
deleteF	The delete function associated with this UDA object.
controlF	The control function associated with this UDA object.

## UDAFilterType

The UDAFilterType represents UDA Filters, which take input from a UDA Reader or UDA Filter, perform some encoding or decoding operation, and output the data to a memory buffer.

```
typedef struct UDAFilterType {
    UInt16          optionFlags;
    UDADeleteFunction deleteF;
    UDAControlFunction controlF;
    UDAReadFunction   readF;
    UDARouterType*    upperReader;
} UDAFilterType
```

### Field Descriptions

optionFlags	Options for the object. This is a combination of values described in <a href="#">Object Option Flags</a> .
deleteF	The delete function associated with this UDA object.
controlF	The control function associated with this UDA object.
readF	The read function associated with this UDA object.
upperReader	The <a href="#">UDAReadertype</a> or <a href="#">UDAFilterType</a> object that reads the data that this object outputs.

## UDAReadertype

The UDAReadertype represents UDA Readers, which read input from a medium.

```
typedef struct UDAReadertag {
    UInt16          optionFlags;
    UDADeleteFunction  deleteF;
    UDAControlFunction  controlF;
    UDAReadFunction   readF;
} UDAReadertype
```

### Field Descriptions

optionFlags	Options for the object. This is a combination of values described in <a href="#">Object Option Flags</a> .
deleteF	The delete function associated with this UDA object.

## Unified Data Access Manager

### UDA Manager Data Structures

---

controlF	The control function associated with this UDA object.
readF	The read function associated with this UDA object.

## UDAWriterType

The UDAWriterType represents UDA Writers, which take data from a UDA Reader or UDA Filter and write the data to an output medium.

```
typedef struct UDAWriterTag {
    UInt16          optionFlags;
    UDADeleteFunction deleteF;
    UDAControlFunction controlF;
    UDAWriteFunction  initiateWriteF;
    UDAFlushFunction   flushF;
    UDAREaderType*    upperReader;
} UDAWriterType
```

### Field Descriptions

optionFlags	Options for the object. This is a combination of values described in <a href="#">Object Option Flags</a> .
deleteF	The delete function associated with this UDA object.
controlF	The control function associated with this UDA object.
initiateWriteF	The write function associated with this UDA object.
flushF	The flush function associated with this UDA object.
upperReader	The <a href="#">UDAREaderType</a> object that reads the data that this object writes.

## UDA Manager Constants

This section describes the constants used with the UDA Manager, which include the following constant types:

- [Control Flags](#)
- [Error Constants](#)
- [Object Option Flags](#)
- [Miscellaneous Constants](#)

### Control Flags

Use the control flag constants to control UDA objects with the [UDAControl](#) macro.

Constant	Value	Description
kUDAReinitialize	1	Used with the <a href="#">UDAControl</a> macro to reinitialize the UDA object.

### Error Constants

At the time of this writing, there is only one error constant associated with the UDA object API.

Constant	Description
udaErrControl	Returned by the <a href="#">UDAControl</a> macro when the control parameter is not valid for the UDA object.

### Object Option Flags

You use the object option flag constants to determine information about the internal state of UDA objects. Note that the [UDAEndOfReader](#) and [UDAMoreData](#) macros provide you with a convenient means of accessing this same information.

## Unified Data Access Manager

### UDA Manager Functions

---

Constant	Value	Description
kUDAEndOfReader	1	Indicates that the UDA reader has reached the end of its data.
kUDAMoreData	2	Indicates that the UDA reader needs more space in the read buffer to do its work.

## Miscellaneous Constants

Constant	Value	Description
kUDAZeroTerminatedBuffer	0xFFFF	Indicates that the buffer is a null-terminated string. Use this value when creating or reinitializing a UDAMemoryReader object.

## UDA Manager Functions

### UDAControl

**Purpose** Applies controls to a UDA object.

**Declared In** UDAMgr.h

**Prototype** Err UDAControl (UDAObjectType\* ioObjectP,  
UInt16 parameter, va\_args)

**Parameters** -> ioObjectP A pointer to the [UDAObjectType](#) object that you want to control. This can be a [UDAResourceType](#), a [UDAFilterType](#), or a [UDAWriterType](#) object.

-> parameter The control action that you want applied to the object.

-> va\_args      Additional parameters, as required for the control and object type.

**Result**    Returns errNone if no error, or udaErrorClass if the control parameter is not valid for the ioObjectP.

**Comments**    The UDAControl function applies a control action to a UDA object. You may need to supply additional parameters, depending on the object type and control parameter values.  
 The only control action defined in Palm OS 4.0 is kUDAREinitialize. You can use it as shown in [Table 86.1](#).

**Table 86.1 UDAControl actions**

Object Type	Usage	Action
UDAExchangeReaderType	UDAControl (myExgRdr, kUDAREinitialize)	Does nothing
UDAExchangeWriterType	UDAControl (myExgWtr, kUDAREinitialize)	Does nothing
UDAMemoryReaderType	UDAControl (myMemRdr, kUDAREinitialize, bufferP, bufferSize)	Reinstalls a new buffer for the memory reader. See <a href="#">UDAMemoryReaderNe</a> <a href="#">w</a> for more information about the parameters.

**Compatibility**    Implemented only if [4.0 New Feature Set](#) is present.

## Unified Data Access Manager

### UDA Manager Functions

---

## UDADelete

**Purpose** Macro that deletes a UDA object.

**Declared In** UDAMgr.h

**Prototype** UDADelete (ioObjectP)

**Parameters** -> ioObjectP A pointer to the [UDAObjectType](#) object that you want to delete. This can be a [UDAResultType](#), a [UDAFilterType](#), or a [UDAWriterType](#) object.

**Result** Returns nothing.

**Comments** The ioObjectP pointer is not valid after this macro completes.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## UDAEndOfReader

**Purpose** Macro that tests if the end of the reader has been reached.

**Declared In** UDAMgr.h

**Prototype** UDAEndOfReader (ioReaderP)

**Parameters** -> ioReaderP A pointer to a [UDAResultType](#) object.

**Result** Returns true if the end of the reader referenced by ioReaderP has been reached, and false if not.

**Comments** The end of the reader has been reached.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## UDAFilterJoin

**Purpose** Macro that joins a filter with a reader.

**Declared In** UDAMgr.h

**Prototype** UDAFilterJoin (ioFilterP, newReaderP)

**Parameters**

- > ioFilterP A pointer to a [UDAFilterType](#) object.
- > newReaderP A pointer to the [UDAResaderType](#) object with which you want the filter joined.

**Result** Returns nothing.

**Comments** Each [UDAFilterType](#) object receives its data from the [UDAResaderType](#) object to which it is joined; this relationship is established when you create the filter object. You can use this macro to change the reader with which the filter is joined. Upon completion, the filter referenced by ioFilterP is joined with the reader referenced by newReaderP.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## UDAIinitiateWrite

**Purpose** Macro that causes the [UDAWriterType](#) object to read data and then write that data to output.

**Declared In** UDAMgr.h

**Prototype** UDAInitiateWrite (ioWriterP)

**Parameters** -> ioWriterP A pointer to a [UDAWriterType](#) object.

**Result** Returns errNone if successful, and an error code if not.

## Unified Data Access Manager

### UDA Manager Functions

---

<b>Comments</b>	When you use this macro, the <code>ioWriterP</code> reads data from the reader to which it is joined. It reads data until the reader is empty, and then writes the data to the output medium.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.

## UDAMoreData

<b>Purpose</b>	Macro that tests if there is more data available to read, but not enough room in the buffer to read it in.
<b>Declared In</b>	<code>UDAMgr.h</code>
<b>Prototype</b>	<code>UDAMoreData (ioReaderP)</code>
<b>Parameters</b>	<code>-&gt; ioReaderP</code> A pointer to a <a href="#">UDAResaderType</a> object.
<b>Result</b>	Returns <code>true</code> if there is more data available for the reader and <code>false</code> if there is no more data available.
<b>Comments</b>	You can use this macro with <a href="#">UDAResaderType</a> objects to determine if there is more data waiting to read. This can happen when the reader's buffer is full.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.

## UDARead

<b>Purpose</b>	Macro that uses the specified <a href="#">UDAResaderType</a> object to read data from the input source and place that data into the specified buffer.
<b>Declared In</b>	<code>UDAMgr.h</code>
<b>Prototype</b>	<code>UDARead (ioReaderP, bufferToFillP, bufferSize, errorP)</code>
<b>Parameters</b>	<code>-&gt; ioReaderP</code> A pointer to a <a href="#">UDAResaderType</a> object that performs the read.

-> bufferToFillP      A pointer to the buffer into which data is placed.  
-> bufferSize      The size of the buffer, in bytes.  
-> errorP      A pointer to an Err value that represents the result of the read operation; if the operation is successful, the value is set to errNone.

**Result**      Returns the number of bytes that were actually read. This value can be less than or equal to the value of bufferSize.

**Comments**      The reader reads from the input source associated with the reader object and places the data into the specified buffer, reading no more than bufferSize bytes of data.

**Compatibility**      Implemented only if [4.0 New Feature Set](#) is present.

## UDAWriterFlush

**Purpose**      Macro that flushes the buffer used by the [\\_UDAWriterType](#) object.

**Declared In**      UDAMgr.h

**Prototype**      UDAWriterFlush (ioWriterP)

**Parameters**      -> ioWriterP      A pointer to a [\\_UDAWriterType](#) object.

**Result**      Returns errNone if successful, and an error code if not.

**Comments**      You can use this macro to flush any data remaining in the buffer of the writer object referenced by ioWriterP. This causes any data in the buffer to be sent to the output medium.

**Compatibility**      Implemented only if [4.0 New Feature Set](#) is present.

## UDAWriterJoin

**Purpose** Macro that joins a writer object to a different reader object.

**Declared In** UDAMgr.h

**Prototype** UDAWriterJoin (ioWriterP, newReaderP)

**Parameters**

- > ioWriterP A pointer to a [UDAWriterType](#) object.
- > newReaderP A pointer to the [UDAReaderType](#) object with which you want the writer joined.

**Result** Returns nothing.

**Comments** Each [UDAWriterType](#) object receives its data from the [UDAReaderType](#) object to which it is joined; this relationship is established when you create the writer object. You can use this macro to change the reader with which the writer is joined. Upon completion, the writer referenced by ioWriterP is joined with the reader referenced by newReaderP.

**Compatibility** Implemented only if 4.0 Feature Set is present.

## UDA Object Creation Functions

### UDAExchangeReaderNew

<b>Purpose</b>	Creates a new <a href="#">UDAReaderType</a> object that you can use to read data from an Exchange Manager socket.
<b>Declared In</b>	UDAMgr.h
<b>Prototype</b>	UDAReaderType* UDAExchangeReaderNew (ExgSocketPtr socket)
<b>Parameters</b>	-> socket A pointer to an <a href="#">ExgSocketType</a> structure that describes the connection.
<b>Result</b>	Returns a pointer to the newly created <a href="#">UDAReaderType</a> object, or NULL if the reader could not be created.
<b>Comments</b>	Use this function to create a UDA Reader object that takes input from an Exchange Manager socket.
<b>Compatibility</b>	Implemented only if <a href="#">4.0 New Feature Set</a> is present.
<b>See Also</b>	<a href="#">ExgSocketType</a>

### UDAExchangeWriterNew

<b>Purpose</b>	Creates a new <a href="#">UDAWriterType</a> object that you can use to write data to an Exchange Manager socket.
<b>Declared In</b>	UDAMgr.h
<b>Prototype</b>	UDAWriterType* UDAExchangeWriterNew (ExgSocketPtr socket, UDABufferSize bufferSize)
<b>Parameters</b>	-> socket A pointer to an <a href="#">ExgSocketType</a> structure that describes the connection.

## Unified Data Access Manager

### UDA Object Creation Functions

---

-> bufferSize The size, in bytes, of the buffer for the new writer object.

**Result** Returns a pointer to the newly created UDA Writer, or NULL if the writer could not be created.

**Comments** Use this function to create a UDA Writer object that sends output to an Exchange Manager socket.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

**See Also** [ExgSocketType](#)

## UDAMemoryReaderNew

**Purpose** Creates a new [UDAResaderType](#) object that you can use to read data from a memory buffer.

**Declared In** UDAMgr.h

**Prototype** `UDAResaderType* UDAMemoryReaderNew  
(const UInt8* bufferP,  
UDABufferSize bufferSizeInBytes)`

**Parameters** -> bufferP A pointer to a buffer in memory from which data is read.

-> bufferSize The size of the buffer, in bytes. If this value is equal to kUDAZeroTerminatedBuffer, bufferP must point to a null-terminated string buffer.

**Result** Returns a pointer to the newly created [UDAResaderType](#) object, or NULL if the reader could not be created.

**Comments** Use this function to create a reader that takes input from memory.

**Compatibility** Implemented only if [4.0 New Feature Set](#) is present.

## **Unified Data Access Manager**

### *UDA Object Creation Functions*

---

# **Part V: Appendixes**



# A

# System Use Only Functions

---

This appendix lists functions that are purposely undocumented because they are for system use only.

---

**WARNING!** System Use Only.

---

AbtShowAbout  
AlmAlarmCallback  
AlmCancelAll  
AlmDisplayAlarm  
AlmEnableNotification  
AlmInit  
AlmTimeChange  
BtLibHandleEvent  
BtLibHandleTransportEvent  
BtLibMutex  
BtLibOpenBackground  
BtLibServiceClose  
BtLibServiceIndicateSessionStart  
BtLibServiceOpen  
BtLibServicePlaySound  
BtLibSleep  
BtLibUnload  
BtLibWake  
Crc16CalcBigBlock  
DmInit  
DmResetRecordStates  
ErrAlertCustom  
EvtDequeueKeyEvent  
EvtEnqueuePenPoint  
EvtGetSysEvent  
EvtInitialize

## System Use Only Functions

---

EvtSetKeyQueuePtr  
EvtSetPenQueuePtr  
EvtSysInit  
ExgInit  
ExgNotifyReceiveV35  
ExpCardInserted  
ExpCardRemoved  
ExpInit  
ExpSlotRegister  
ExpSlotUnregister  
FileReadLow  
Find  
FrmActiveState  
FrmAddSpaceForObject  
Frm GetUserModifiedState  
FrmSetNotUserModified  
FtrInit  
GrfFieldChange  
GrfFree  
GrfInit  
HostControl  
INetLibSleep  
INetLibSockMailAttrGet  
INetLibSockMailAttrSet  
INetLibSockMailQueryProgress  
INetLibSockMailReqAdd  
INetLibSockMailReqCreate  
INetLibSockMailReqSend  
INetLibWake  
InsPtCheckBlink  
InsPtInitialize  
IntlInit  
IrHandleEvent  
IrWaitForEvent  
KeyboardStatusNew  
KeyboardStatusFree  
KbdsetLayout  
KbdGetLayout  
KbdsetPosition  
KbdGetPosition

KbdSetShiftState  
KbdGetShiftState  
KbdDraw  
KbdErase  
KbdHandleEvent  
MemCardFormat  
MemChunkFree  
MemChunkNew  
MemHandleFlags  
MemHandleLockCount  
MemHandleOwner  
MemHandleResetLock  
MemHeapFreeByOwnerID  
MemHeapInit  
MemInit  
MemInitHeapTable  
MemKernelInit  
MemPtrFlags  
MemPtrOwner  
MemPtrResetLock  
MemSemaphoreRelease  
MemSemaphoreReserve  
MemStoreSetInfo  
NetLibConfigAliasGet  
NetLibConfigAliasSet  
NetLibConfigDelete  
NetLibConfigIndexFromName  
NetLibConfigList  
NetLibConfigMakeActive  
NetLibConfigRename  
NetLibConfigSaveAs  
NetLibHandlePowerOff  
NetLibOpenConfig  
NetLibOpenIfCloseWait  
NetLibSleep  
NetLibWake  
PenClose  
PenGetRawPen  
PenOpen  
PenRawToScreen

## **System Use Only Functions**

---

PenScreenToRaw  
PenSleep  
PenWake  
ResLoadString  
ScrCompressScanLine  
ScrCopyRectangle  
ScrDeCompressScanLine  
ScrDrawChars  
ScrDrawNotify  
ScrInit  
ScrLineRoutine  
ScrRectangleRoutine  
ScrScreenInfo  
ScrSendUpdateArea  
SerDbgAssureOpen  
SerialMgrInstall  
SerReceiveISP  
SerReceiveWindowClose  
SerReceiveWindowOpen  
SerSetMapPort  
SerSetWakeupHandler  
SerSleep  
SerWake  
SlkProcessRPC  
SlkSysPktDefaultResponse  
SndInit  
SndSetDefaultVolume  
SrmSelectorErrPrv  
SrmSleep  
SrmWake  
SysAppStartup  
SysAppExit  
SysBatteryDialog  
SysCardImageDeleted  
SysCardImageInfo  
SysColdBoot  
SysDisableInts  
SysDoze  
SysEvGroupCreate  
SysEvGroupRead

SysEvGroupSignal  
SysEvGroupWait  
SysInit  
SysKernelInfo  
SysLaunchConsole  
SysLCDBrightness  
SysLCDContrast  
SysLibClose  
SysLibInstall  
SysLibOpen  
SysLibSleep  
SysLibTblEntry  
SysLibWake  
SysMailboxCreate  
SysMailboxDelete  
SysMailboxFlush  
SysMailboxSend  
SysMailboxWait  
SysNewOwnerID  
SysPowerOn  
SysResSemaphoreCreate  
SysResSemaphoreDelete  
SysResSemaphoreRelease  
SysResSemaphoreReserve  
SysRestoreStatus  
SysSemaphoreCreate  
SysSemaphoreDelete  
SysSemaphoreSet  
SysSemaphoreSignal  
SysSemaphoreWait  
SysSetA5  
SysSetPerformance  
SysSleep  
SysTaskCreate  
SysTaskDelete  
SysTaskID  
SysTaskResume  
SysTaskSetTermProc  
SysTaskSuspend  
SysTaskSwitching

## **System Use Only Functions**

---

SysTaskTrigger  
SysTaskWait  
SysTaskWaitClr  
SysTaskWake  
SysTimerCreate  
SysTimerDelete  
SysTimerRead  
SysTimerWrite  
SysTranslateKernelErr  
SysUIBusy  
SysUILaunch  
SysUnimplemented  
SysWantEvent  
TimInit  
TxtPrepFindString  
UIColorPopTable  
UIColorPushTable  
UIInitialize  
UIReset  
UIPopTable  
UIPushTable  
VFSInit  
WinAddWindow  
WinDrawWindowFrame  
WinDisableWindow  
WinEnableWindow  
WinGetWindowPointer  
WinInitializeWindow  
WinMoveWindowAddr  
WinRemoveWindow  
WinScreenInit  
WinSetColors

# Compatibility Guide

---

This appendix lists groups of functions and other features (such as events and launch codes) that have been added to the Palm OS® after version 1.0.

Before you use any new functions or features in an application, you must check to ensure that they are implemented in the OS version your application is running on. Checking the OS version number is **not** a reliable indicator that a specific feature is present, since some later OS versions do not include features present in earlier versions. In order to ensure that your code is supported, you must check for the presence of individual features.

To make this check easier, this appendix lists new functions and features in groups such that all functions and features in a group are always implemented together in the ROM of a Palm™ device. This means that you can check for a single feature in that group and be assured that if that feature is present than all functions and features in that group are implemented.

Each group includes a recommended test to check if it is implemented. The following groups are described:

- [2.0 New Feature Set](#)
- [3.0 New Feature Set](#)
- [3.1 New Feature Set](#)
- [3.2 New Feature Set](#)
- [International Feature Set](#)
- [Japanese Feature Set](#)
- [Wireless Internet Feature Set](#)
- [New Serial Manager Feature Set](#)
- [Connection Manager Feature Set](#)
- [3.5 New Feature Set](#)
- [Notification Feature Set](#)

## Compatibility Guide

### 2.0 New Feature Set

---

- [4.0 New Feature Set](#)
- [Expansion Manager Feature Set](#)
- [VFS Manager Feature Set](#)
- [Bluetooth Library Feature Set](#)
- [High-Density Display Feature Set](#)
- [Sound Stream Feature Set](#)
- [5.0 New Feature Set](#)
- [5.1 New Feature Set](#)

## 2.0 New Feature Set

You can check that this feature set is implemented by checking that the system version is 2.0 or higher. Use this [FtrGet](#) call:

```
err = FtrGet(sysFtrCreator,  
    sysFtrNumROMVersion, &romversion);
```

The romversion parameter should be 0x02003000 or greater.

### Launch Codes

This feature set adds the following launch codes:

[sysAppLaunchCmdLookup](#)  
[sysAppLaunchCmdPanelCalledFromApp](#)  
[sysAppLaunchCmdReturnFromPanel](#)  
[sysAppLaunchCmdSystemLock](#)

### Functions

This feature set adds the following functions:

<a href="#">CategoryInitialize</a>	<a href="#">CategorySetName</a>
<a href="#">DmDeleteCategory</a>	<a href="#">DmDatabaseProtect</a>
<a href="#">EvtAddUniqueEventToQueue</a>	<a href="#">EvtEventAvail</a>
<a href="#">EvtSysEventAvail</a>	

<a href="#">FldGetNumberOfBlankLines</a>	<a href="#">FldGetScrollValues</a>
<a href="#">FldSetInsertionPoint</a>	
<a href="#">FntGetScrollValues</a>	<a href="#">FntWordWrap</a>
<a href="#">FntWordWrapReverseNLines</a>	
<a href="#">FrmPointInTitle</a>	<a href="#">FrmSetMenu</a>
<a href="#">FrmSetObjectBounds</a>	
<a href="#">KeySetMask</a>	<a href="#">LocGetNumberSeparator</a>
	<a href="#">S</a>
<a href="#">LstScrollList</a>	<a href="#">LstGetVisibleItems</a>
<a href="#">MemCmp</a>	<a href="#">MenuSetActiveMenuRscID</a>
<a href="#">PhoneNumberLookup</a>	
<a href="#">PrefSetPreference</a>	<a href="#">PrefGetPreference</a>
<a href="#">SclDrawScrollBar</a>	<a href="#">SclGetScrollBar</a>
<a href="#">SclHandleEvent</a>	<a href="#">SclSetScrollBar</a>
<a href="#">SerControl</a>	
<a href="#">StrDelocalizeNumber</a>	<a href="#">StrLocalizeNumber</a>
<a href="#">StrNCaselessCompare</a>	<a href="#">StrNCat</a>
<a href="#">StrNCompare</a>	<a href="#">StrNCopy</a>
<a href="#">StrPrintF</a>	<a href="#">StrVPrintF</a>
<a href="#">SysBinarySearch</a>	<a href="#">SysCreateDataBaseList</a>
<a href="#">SysCreatePanelList</a>	<a href="#">SysErrString</a>
<a href="#">SysGraffitiReferenceDialog</a>	<a href="#">SysLibLoad</a>
<a href="#">SysStringByIndex</a>	<a href="#">SysTicksPerSecond</a>
<a href="#">TblHasScrollBar</a>	<a href="#">TblSetBounds</a>
<a href="#">TblSetColumnEditIndicator</a>	<a href="#">TblSetRowStaticHeight</a>
WinSetWindowBounds	

## Existing Functions that Changed

Several functions that existed in 1.0 were changed in 2.0:

## Compatibility Guide

2.0 New Feature Set

---

[CategoryCreateList](#) (old function renamed  
[CategoryCreateListV10](#))

[CategoryEdit](#) (old function renamed [CategoryEditV10](#))

[CategoryFreeList](#) (old function renamed  
[CategoryFreeListV10](#))

[CategorySelect](#) (old function renamed [CategorySelectV10](#))

[SelectDay](#) (old function renamed [SelectDayV10](#))

[DmFindSortPosition](#) (old function renamed  
[DmFindSortPositionV10](#))

[PrefGetAppPreferences](#) (old function renamed  
[PrefGetAppPreferencesV10](#))

[PrefOpenPreferenceDB](#) (old function renamed  
[PrefOpenPreferenceDBV10](#))

[PrefSetAppPreferences](#) (old function renamed  
[PrefSetAppPreferencesV10](#))

[SerReceive](#) (old function renamed [SerReceiveV10](#))

[SerSend](#) (old function renamed [SerSendV10](#))

[SysKeyboardDialog](#) (old function renamed  
[SysKeyboardDialogV10](#))

## Other Changes

As a rule, all Palm OS applications developed with the 1.0 SDK should run error-free on the latest device. There are two possible pitfalls for 1.0 applications:

- **fldChangedEvent Change**—The operating system now correctly sends a fldChangedEvent whenever a field object is changed. Previously, the event was at times not sent, especially when a FldSetText operation was performed. If your application doesn't catch the events that are now sent, it may have problems.
- **Non-standard tools**—If your application was not developed with Metrowerks Code Warrior for the Palm OS, it may run

into problems. One known problem can occur if the application:

- was compiled with optimization turned on
- uses system preferences

## Compatibility Guide

### 3.0 New Feature Set

---

## 3.0 New Feature Set

You can check that this feature set is implemented by checking that the system version is 3.0 or higher. Use this [FtrGet](#) call:

```
err = FtrGet(sysFtrCreator,  
    sysFtrNumROMVersion, &romVersion);
```

The romVersion parameter should be greater than or equal to 0x03003000, which can be constructed using the [sysMakeROMVersion](#) macro:

```
sysMakeROMVersion(3, 0, 0, sysROMStageRelease, 0)
```

## Launch Codes

This feature set adds the following launch codes:

[sysAppLaunchCmdExgAskUser](#)  
[sysAppLaunchCmdExgReceiveData](#)

In addition, the launch code [sysAppLaunchCmdGoto](#) is now also sent by the exchange manager, in addition to its use by the global find operation.

## Font

This feature set adds the following font:

largeBoldFont

## Functions

This feature set adds the following functions:

### Dynamic User Interface Functions

<a href="#">CtlNewControl</a>	<a href="#">FrmNewLabel</a>
<a href="#">CtlValidatePointer</a>	<a href="#">FrmRemoveObject</a>
<a href="#">FldNewField</a>	<a href="#">FrmValidatePtr</a>
<a href="#">FrmNewBitmap</a>	<a href="#">LstNewList</a>
<a href="#">FrmNewForm</a>	<a href="#">WinValidateHandle</a>
<a href="#">FrmNewGadget</a>	

For more information on creating and using dynamic user interface elements, see the section “[Dynamic UI](#)” on page 142 of the *Palm OS Programmer’s Companion*, vol. I.

## Font Functions

[FontSelect](#)  
[FntDefineFont](#)

For more information on these functions and the support for custom fonts, see “[Fonts](#)” on page 268 of the *Palm OS Programmer’s Companion*, vol. I.

## Progress Manager Functions

[PrgHandleEvent](#)                    [PrgUpdateDialog](#)  
[PrgStartDialog](#)                    [PrgUserCancel](#)  
[PrgStopDialog](#)

For more information, see the section “[Progress Dialogs](#)” on page 86 of the *Palm OS Programmer’s Companion*, vol. I.

## File Streaming Functions

<a href="#">FileClearerr</a>	<a href="#">FileOpen</a>
<a href="#">FileClose</a>	<a href="#">FileRead</a>
<a href="#">FileControl</a>	FileReadLow (system use only)
<a href="#">FileDelete</a>	<a href="#">FileRewind</a>
<a href="#">FileDmRead</a>	<a href="#">FileSeek</a>
<a href="#">FileEOF</a>	<a href="#">FileTell</a>
<a href="#">FileError</a>	<a href="#">FileTruncate</a>
<a href="#">FileFlush</a>	<a href="#">FileWrite</a>
<a href="#">FileGetLastError</a>	

For more information, see the section “[File Streaming Application Program Interface](#)” on page 202 of the *Palm OS Programmer’s Companion*, vol. I.

## Sound Functions

[SndCreateMidiList](#)  
[SndPlaySmf](#)  
[SndDoCmd](#) (enhanced in 3.0)

## Compatibility Guide

3.0 New Feature Set

---

### Exchange Manager Functions

[ExgAccept](#)

[ExgDBRead](#)

[ExgDBWrite](#)

[ExgDisconnect](#)

[ExgPut](#)

[ExgReceive](#)

[ExgRegisterData](#)

[ExgSend](#)

For more information, see the chapter [Beaming \(Infrared Communication\)](#) in the *Palm OS Programmer's Companion*, vol. II, *Communications*.

### IR Library Functions

[IrAdvanceCredit](#)

[IrBind](#)

[IrClose](#)

[IrConnectIrLap](#)

[IrConnectReq](#)

[IrConnectRsp](#)

[IrDataReq](#)

[IrDisconnectIrLap](#)

[IrDiscoverReq](#)

[IrIAS\\_Add](#)

[IrIAS\\_GetInteger](#)

[IrIAS.GetIntLsap](#)

[IrIAS\\_GetObjectID](#)

[IrIAS\\_GetOctetString](#)

[IrIAS\\_GetOctetStringLen](#)

[IrIAS.GetType](#)

[IrIAS.GetUserString](#)

[IrIAS.GetUserStringCharSet](#)

[IrIAS.GetUserStringLen](#)

[IrIAS\\_Next](#)

[IrIAS\\_Query](#)

[IrIAS\\_SetDeviceName](#)

[IrIAS\\_StartResult](#)

[IrIsIrLapConnected](#)

[IrIsMediaBusy](#)

[IrIsNoProgress](#)

[IrIsRemoteBusy](#)

[IrLocalBusy](#)

[IrMaxRxSize](#)

[IrMaxTxSize](#)

[IrOpen](#)

[IrSetConTypeLMP](#)

[IrSetConTypeTTP](#)

[IrSetDeviceInfo](#)

[IrTestReq](#)

[IrUnbind](#)

For more information, see the chapter [Beaming \(Infrared Communication\)](#) in the *Palm OS Programmer's Companion*, vol. II, *Communications*.

## Miscellaneous Functions

<a href="#">FrmRestoreActiveState</a>	<a href="#">SysGetROMToken</a>
<a href="#">FrmSaveActiveState</a>	<a href="#">SysGetStackInfo</a>
<a href="#">ScrDisplayMode</a>	<a href="#">SysGremlins</a>
<a href="#">SysGetAppInfo</a> (system use only)	<a href="#">TblGetItemFont</a>
<a href="#">SysGetOSVersionString</a>	<a href="#">TblSetItemFont</a>

## Existing Functions that Changed

Two functions that existed in 2.0 were changed in 3.0:

[CategoryEdit](#) (old function renamed [CategoryEditV20](#))

[SysBatteryInfo](#) (old function renamed [SysBatteryInfoV20](#))

## Other Changes

- The dynamic heap has been increased in size to 96 KB.
- Storage RAM is no longer subdivided into multiple storage heaps of 64 KB each. All storage RAM on a memory card is configured as a single storage heap.
- Each flash ROM-based Palm device holds a serial number that identifies it uniquely and can be retrieved via [SysGetROMToken](#). For more information, see “[Retrieving the ROM Serial Number](#)” on page 353 of the *Palm OS Programmer’s Companion*, vol. I.
- The Application Launcher (accessed via the silkscreen “Applications” button) is now an application, rather than a popup. The `SysAppLauncherDialog` function, which provides the API to the old popup launcher, is still present in Palm OS 3.0 for compatibility purposes, but has not been updated and generally should not be used. For more information, see “[Application Launcher](#)” on page 153 of the *Palm OS Programmer’s Companion*, vol. I.
- The sound manager supports MIDI sound files, adding new sounds, asynchronous playback, and other features. There are also new selectors for setting the volume preferences. For more information, see the section “[System Boot and Reset](#)” on page 337 of the *Palm OS Programmer’s Companion*, vol. I.

## Compatibility Guide

### 3.1 New Feature Set

---

The following functions existed in the system previously, but were not documented:

[RctCopyRectangle](#)  
[RctGetIntersection](#)  
[RctInsetRectangle](#)

[RctOffsetRectangle](#)  
[RctPtInRectangle](#)  
[RctSetRectangle](#)

The following event type existed in the system previously, but was not previously documented:

[frmGotoEvent](#)

## 3.1 New Feature Set

You can check that this feature set is implemented by checking that the system version is 3.1 or higher. Use this [FtrGet](#) call:

```
err = FtrGet(sysFtrCreator,  
             sysFtrNumROMVersion, &romVersion);
```

The romVersion parameter should be greater than or equal to 0x03103000, which can be constructed using the [sysMakeROMVersion](#) macro:

```
sysMakeROMVersion(3, 1, 0, sysROMStageRelease, 0)
```

## Functions

This feature set adds the following functions:

[ChrHorizEllipsis](#)  
[ChrNumericSpace](#)  
[ContrastAdjust](#)  
[FntWidthToOffset](#)  
[FtrPtrNew](#)  
[FtrPtrFree](#)  
[FtrPtrResize](#)  
[SelectOneTime](#)  
[WinDrawChar](#)  
[WinDrawTruncChars](#)

---

**NOTE:** The `PalmOSGlue.lib` provides compatibility functions and macros for `ChrHorizEllipsis`, `ChrNumericSpace`, `WinDrawChar`, and `WinDrawTruncChars`. If you want to use these functions on systems that don't have the 3.1 feature set, you can link your application with `PalmOSGlue.lib`. See the chapter "[PalmOSGlue Library](#)" on page 1891 for more information.

---

## Changes to the Character Encoding

Starting in Palm OS 3.1, the character encoding used on most systems is Microsoft Windows code page 1252. Versions prior to 3.1 used an encoding that was very similar to code page 1252 but did not follow it exactly. The following changes to the character set are introduced in Palm OS 3.1:

- Some of the special Palm OS glyphs in the high ASCII range (such as the shortcut stroke and the command stroke) have been moved down into the control code range, and other characters (such as the numeric space and horizontal ellipsis) have been copied into the control range so that they're guaranteed to exist in every encoding. For the numeric space and horizontal ellipsis, you can use the macros `ChrNumericSpace` and `ChrHorizEllipsis` to return the appropriate character regardless of the character map. In `PalmOSGlue.lib`, these two macros are named `TxtGlueGetNumericSpaceChar` and `TxtGlueGetHorizEllipsisChar`, respectively.
- The four playing-card characters have been moved from the high ASCII range in the standard four fonts to the 9-point Symbol font.

## Other Changes in 3.1

- Palm OS 3.1 supports a new processor: the EZ Dragonball processor. This processor is compatible with the existing Dragonball processor, so your application should run

## Compatibility Guide

### 3.1 New Feature Set

---

without changes as long as it doesn't access registers or system globals directly.

If your application needs to know if it is running on an EZ Dragonball, it can check using the following code:

```
DWord id, chip;
Word revision;
Err err;
err = FtrGet(sysFtrCreator,
              sysFtrNumProcessorID, &id);
if (!err) {
    chip = id & sysFtrNumProcessorMask;
    revision = id & 0xffff;
    if (chip==sysFtrNumProcessor328)
        // traditional Dragonball
    else if (chip==sysFtrNumProcessorEZ)
        // Dragonball EZ
}
```

- The constant preferenceDataVersion was removed and replaced with preferenceDataVerLatest.
- Character variables are now two bytes long. The type WChar defines a character variable.
- The keyDownEvent structure's chr field (which contains the input character) has been changed from a Word to a WChar.
- The string manager functions [StrChr](#) and [StrStr](#) now treat buffers as characters, not arbitrary byte arrays. If you previously used these functions to search data buffers, your code may no longer work.
- The string manager function [StrToLower](#) can now handle any type of characters, including accented characters.
- The underline attribute of [FieldAttrType](#) now has support for the value 2. Previously, the only underline modes available were no underline (0) and gray underline (1). In Palm OS 3.1 and higher, the value 2 is interpreted as solid underline. The UnderlineModeType enum defined in Window.h defines the possible values for the underline attribute.

- The use of the [DmGetNextDatabaseByTypeCreator](#) `onlyLatestVers` parameter changed in 3.1. If `onlyLatestVers` is true, you only receive one matching database for each type/creator pair. In version 3.0 and earlier, you could receive multiple matching databases if `onlyLatestVers` was true. See that function's description for a more detailed description.

## 3.2 New Feature Set

You can check that this feature set is implemented by checking that the system version is 3.2 or higher. Use this [FtrGet](#) call:

```
err = FtrGet(sysFtrCreator,  
             sysFtrNumROMVersion, &romVersion);
```

The `romVersion` parameter should be greater than or equal to `0x03203000`, which can be constructed using the [sysMakeROMVersion](#) macro:

```
sysMakeROMVersion(3, 2, 0, sysROMStageRelease, 0)
```

### Functions

This feature set adds the following functions:

<a href="#">AlmGetProcAlarm</a>	<a href="#">NetLibConfigIndexFromName</a>
<a href="#">AlmSetProcAlarm</a>	<a href="#">NetLibConfigList</a>
<a href="#">ClipboardAppendItem</a>	<a href="#">NetLibConfigMakeActive</a>
<a href="#">DmGetDatabaseLockState</a>	<a href="#">NetLibConfigRename</a>
<a href="#">ErrAlert</a>	<a href="#">NetLibConfigSaveAs</a>
<a href="#">NetLibConfigAliasGet</a>	<a href="#">NetLibOpenConfig</a>
<a href="#">NetLibConfigAliasSet</a>	<a href="#">SndPlaySmfResource</a>
<a href="#">NetLibConfigDelete</a>	

Note that the `NetLib...` functions, although present in the 3.2 New Feature Set, are first declared in the Palm OS 5 SDK.

## Existing Functions that Changed

Two functions that existed in 3.0 were changed in 3.2:

[SysGremlins](#) was removed and replaced with a SysGremlins macro that maps it to the function HostGremlinIsRunning. The prototype is slightly different, but you can still call SysGremlins in the same way you did before.

[PrgStartDialog](#) (old function renamed [PrgStartDialogV31](#))

## Other Changes in 3.2

- The prototype for the system use only function AlmDisplayAlarm changed from no return value to a Boolean return value. This change may affect system patches and extensions that intercept AlmDisplayAlarm calls.

## International Feature Set

You can check that this feature set is implemented by checking for the existence of the international manager. You can check by calling [FtrGet](#) as follows:

```
err = FtrGet(sysFtrCreator, sysFtrNumIntlMgr,  
&value);
```

If the international manager is installed, the value parameter will be non-zero and the returned error should also be zero (for no error).

You can learn more about the international manager by reading the chapter “[Localized Applications](#)” on page 363 in the *Palm OS Programmer’s Companion*, vol. I.

---

**NOTE:** If you want to use international functions on systems that don’t have the international feature, you can link your application with `PalmOSGlue.lib`. The functions in this library are the same as those listed below except that they use the prefix “TxtGlue” instead of “Txt.” For more information, see the chapter “[PalmOSGlue Library](#)” on page 1891.

---

## Functions

This feature set adds the following functions:

### Text Manager Functions

<a href="#">TxtByteAttr</a>	<a href="#">TxtCharXAttr</a>
<a href="#">TxtCaselessCompare</a>	<a href="#">TxtCompare</a>
<a href="#">TxtCharAttr</a>	<a href="#">TxtEncodingName</a>
<a href="#">TxtCharBounds</a>	<a href="#">TxtFindString</a>
<a href="#">TxtCharEncoding</a>	<a href="#">TxtGetChar</a>
<a href="#">TxtCharIsAlNum</a>	<a href="#">TxtGetNextChar</a>
<a href="#">TxtCharIsAlpha</a>	<a href="#">TxtGetPreviousChar</a>
<a href="#">TxtCharIsCntrl</a>	<a href="#">TxtCharIsValid</a>
<a href="#">TxtCharIsDigit</a>	<a href="#">TxtMaxEncoding</a>
<a href="#">TxtCharIsGraph</a>	<a href="#">TxtNextCharSize</a>
<a href="#">TxtCharIsHardKey</a>	<a href="#">TxtPreviousCharSize</a>
<a href="#">TxtCharIsHex</a>	<a href="#">TxtReplaceStr</a>
<a href="#">TxtCharIsLower</a>	<a href="#">TxtSetNextChar</a>
<a href="#">TxtCharIsPrint</a>	<a href="#">TxtStrEncoding</a>
<a href="#">TxtCharIsPunct</a>	<a href="#">TxtTransliterate</a>
<a href="#">TxtCharIsSpace</a>	<a href="#">TxtGetTruncationOffset</a>
<a href="#">TxtCharIsUpper</a>	<a href="#">TxtWordBounds</a>
<a href="#">TxtCharSize</a>	

### Other Functions

[IntlGetProcAddress](#)

### Removed Functions and Macros

If the international feature set exists, then the following functions and macros are no longer available:

## Compatibility Guide

### Japanese Feature Set

---

[GetCharAttr](#)  
[GetCharCaselessValue](#)  
[GetCharSortValue](#)  
IsAscii  
IsAlNum  
IsAlpha  
IsCntrl  
IsDigit  
IsGraph  
IsLower  
IsPrint  
IsPunct  
IsSpace  
IsUpper  
IsHex  
IsDelim

## Japanese Feature Set

You can check that the Japanese feature set is implemented by checking if the unit is Japanese. You can check by calling [FtrGet](#) as follows:

```
err = FtrGet (sysFtrCreator, sysFtrNumEncoding,  
&value);
```

The unit has the Japanese OS if the value parameter is charEncodingCP932.

For further information about the Japanese implementation, see the section "[Notes on the Japanese Implementation](#)" in the *Palm OS Programmer's Companion*, vol. I.

## Wireless Internet Feature Set

You can check that this feature set is implemented by checking for the existence of the Web Clipping Application Viewer (*Viewer*) and iMessenger™ applications. Here's an example of how to check for *Viewer*:

```
DmSearchStateType searchState;
UInt16 cardNo;
LocalID dbID;
err = DmGetNextDatabaseByTypeCreator(true,
&searchState, sysFileTApplication,
sysFileCClipper, true, &cardNo, &dbID);
```

If Viewer is not present, the  
DmGetNextDatabaseByTypeCreator routine returns an error.  
To check for iMessenger, you can use the creator type  
sysFileCMessaging.

---

**NOTE:** The Viewer was formerly described as the Clipper.

---

You can learn more about the Palm.Net™ system for wireless Internet access and the programmatic interfaces to the Viewer and iMessenger applications by reading the chapter “[Internet and Messaging Applications](#)” in the *Palm OS Programmer’s Companion*, vol. II, *Communications*. For a more complete description, see the *Web Clipping Developer’s Guide*.

## Launch Codes

This feature set adds the following launch codes:

[sysAppLaunchCmdAddRecord](#) (for iMessenger application; existed for Mail in 3.0)  
[sysAppLaunchCmdGoToURL](#)  
[sysAppLaunchCmdOpenDB](#)  
[sysAppLaunchCmdURLParams](#)

## Events

This feature set adds the following events:

[inetSockReadyEvent](#)  
[inetSockStatusChangeEvent](#)

This feature set also adds the following keyDownEvent key codes:

## Compatibility Guide

### New Serial Manager Feature Set

---

vchrHardAntenna  
vchrRadioCoverageOK  
vchrRadioCoverageFail

These key codes are described in the section [Wireless keyDownEvent Key Codes](#).

## Functions

This feature set adds the following functions.

### Internet Library Functions

For more information, see the chapter "[Network Communication](#)" in the *Palm OS Programmer's Companion*, vol. II, *Communications*.

<a href="#">INetLibCacheGetObject</a>	<a href="#">INetLibSockClose</a>
<a href="#">INetLibCacheList</a>	<a href="#">INetLibSockConnect</a>
<a href="#">INetLibCheckAntennaState</a>	<a href="#">INetLibSockHTTPAttrGet</a>
<a href="#">INetLibClose</a>	<a href="#">INetLibSockHTTPAttrSet</a>
<a href="#">INetLibConfigAliasGet</a>	<a href="#">INetLibSockHTTPReqCreate</a>
<a href="#">INetLibConfigAliasSet</a>	<a href="#">INetLibSockHTTPReqSend</a>
<a href="#">INetLibConfigDelete</a>	<a href="#">INetLibSockOpen</a>
<a href="#">INetLibConfigIndexFromName</a>	<a href="#">INetLibSockRead</a>
<a href="#">INetLibConfigList</a>	<a href="#">INetLibSettingGet</a>
<a href="#">INetLibConfigMakeActive</a>	<a href="#">INetLibSettingSet</a>
<a href="#">INetLibConfigRename</a>	<a href="#">INetLibSockStatus</a>
<a href="#">INetLibConfigSaveAs</a>	<a href="#">INetLibURLCrack</a>
<a href="#">INetLibGetEvent</a>	<a href="#">INetLibURLGetInfo</a>
<a href="#">INetLibOpen</a>	<a href="#">INetLibURLOpen</a>
<a href="#">INetLibSettingGet</a>	<a href="#">INetLibURLsAdd</a>
<a href="#">INetLibSettingSet</a>	<a href="#">INetLibWiCmd</a>

## New Serial Manager Feature Set

The New Serial Manager feature set has two different versions.

## New Serial Manager Feature Set Version 1

You can check that this feature set is implemented by checking for the existence of the new Serial Manager. You can check by calling [FtrGet](#) as follows:

```
err = FtrGet(sysFileCSerialMgr,  
    sysFtrNewSerialPresent, &value);
```

If the new Serial Manager is installed, the value parameter will be non-zero and the returned error should also be zero (for no error).

You can learn more about the new Serial Manager and Connection Manager by reading the sections “[The Serial Manager](#)” on page 92 and “[The Connection Manager](#)” on page 116 in the *Palm OS Programmer’s Companion*, vol. II, *Communications*.

This feature set adds the following functions.

### Serial Manager Functions

<a href="#">SrmClearErr</a>	<a href="#">SrmReceiveFlush</a>
<a href="#">SrmClose</a>	<a href="#">SrmReceiveWait</a>
<a href="#">SrmControl</a>	<a href="#">SrmReceiveWindowClose</a>
<a href="#">SrmGetDeviceCount</a>	<a href="#">SrmReceiveWindowOpen</a>
<a href="#">SrmGetDeviceInfo</a>	<a href="#">SrmSend</a>
<a href="#">SrmGetStatus</a>	<a href="#">SrmSendCheck</a>
<a href="#">SrmOpen</a>	<a href="#">SrmSendFlush</a>
<a href="#">SrmOpenBackground</a>	<a href="#">SrmSendWait</a>
<a href="#">SrmPrimeWakeUpHandler</a>	<a href="#">SrmSetReceiveBuffer</a>
<a href="#">SrmReceive</a>	<a href="#">SrmSetWakeUpHandler</a>
<a href="#">SrmReceiveCheck</a>	<a href="#">WakeUpHandlerProcPtr</a>

### Virtual Driver Functions

<a href="#">DrvEntryPointProcPtr</a>	<a href="#">VdrvStatusProcPtr</a>
<a href="#">GetSizeProcPtr</a>	<a href="#">VdrvWriteProcPtr</a>
<a href="#">GetSpaceProcPtr</a>	<a href="#">WriteBlockProcPtr</a>
<a href="#">VdrvControlProcPtr</a>	<a href="#">WriteByteProcPtr</a>
<a href="#">VdrvOpenProcPtr</a>	

### **Connection Manager Functions**

[CncAddProfile](#)  
[CncDeleteProfile](#)

[CncGetProfileInfo](#)  
[CncGetProfileList](#)

### **Serial Link Manager Function**

[SlkSocketPortID](#)

## **New Serial Manager Feature Set Version 2**

You can check that version 2 of the new Serial Manager feature set is implemented by checking the Serial Manager version number and the Palm OS version number. You can check by calling [FtrGet](#) as follows:

```
err = FtrGet(sysFileCSerialMgr,  
             sysFtrNewSerialVersion, &value);  
err = FtrGet(sysFtrCreator,  
             sysFtrNumROMVersion, &romVersion);
```

The new Serial Manager is present if:

- Both calls to [FtrGet](#) return zero (for no error).
- The [value](#) parameter is 2.
- The [romVersion](#) parameter is 0x04003000, which can be constructed using the [sysMakeROMVersion](#) macro:

```
sysMakeROMVersion(4, 0, 0, sysROMStageRelease, 0)
```

This feature set adds the following functions.

### **Serial Manager Functions**

[SrmCustomControl](#)  
[SrmExtOpenBackground](#)

[SrmExtOpen](#)

### **Virtual Driver Functions**

[VdrvControlCustomProcPtr](#) [VdrvOpenProcV4Ptr](#)  
[SignalCheckPtr](#)

**IMPORTANT:** Some Handspring devices ship with Palm OS version 3.5 but have new Serial Manager feature set version 2. These devices support the `SignalCheckPtr` virtual driver function and have expanded functionality for USB support, but they do not support the other function calls in this feature set.

---

You can learn more about the new Serial Manager by reading the chapter “[Serial Communication](#)” on page 89 of *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## Connection Manager Feature Set

You can check that the Connection Manager feature set is implemented by checking the value of the Connection Manager feature. You can check by calling [`FtrGet`](#) as follows:

```
err = FtrGet (kCncFtrCncMgrCreator,  
              kCncFtrCncMgrVersion, &version);
```

The `version` parameter should be greater than or equal to 0x00040001. In the 4.0 Palm OS SDK, this value is represented by the `kCncMgrVersion` constant.

---

**NOTE:** An earlier version of the Connection Manager is available if [New Serial Manager Feature Set Version 1](#) is present.

---

## Functions

This feature set adds the following functions:

<a href="#">CncProfileCloseDB</a>	<a href="#">CncProfileGetIDFromName</a>
<a href="#">CncProfileCount</a>	<a href="#">CncProfileGetIndex</a>
<a href="#">CncProfileCreate</a>	<a href="#">CncProfileOpenDB</a>
<a href="#">CncProfileDelete</a>	<a href="#">CncProfileSetCurrent</a>
<a href="#">CncProfileGetCurrent</a>	<a href="#">CncProfileSettingGet</a>
<a href="#">CncProfileGetIDFromIndex</a>	<a href="#">CncProfileSettingSet</a>

## Compatibility Guide

### 3.5 New Feature Set

---

## 3.5 New Feature Set

You can check that this feature set is implemented by checking that the system version is 3.5 or higher. Use this [FtrGet](#) call:

```
err = FtrGet(sysFtrCreator,  
             sysFtrNumROMVersion, &romVersion);
```

The romVersion parameter should be greater than or equal to 0x03503000, which can be constructed using the [sysMakeROMVersion](#) macro:

```
sysMakeROMVersion(3, 5, 0, sysROMStageRelease, 0)
```

## Launch Codes

This feature set adds the following launch codes:

[sysAppLaunchCmdNotify](#)

## Events

This feature set adds the following events:

[frmGadgetEnterEvent](#)  
[frmGadgetMiscEvent](#)  
[menuCmdBarOpenEvent](#)  
[menuOpenEvent](#)

## Functions

This feature set adds the following functions.

### Bitmaps

<a href="#">BmpBitsSize</a>	<a href="#">BmpGetBits</a>
<a href="#">BmpColortableSize</a>	<a href="#">BmpGetColortable</a>
<a href="#">BmpCompress</a>	<a href="#">BmpSize</a>
<a href="#">BmpCreate</a>	<a href="#">ColorTableEntries</a>
<a href="#">BmpDelete</a>	

For more information on creating and using bitmaps, see the section “[Bitmaps](#)” on page 123 of the *Palm OS Programmer’s Companion*, vol. I.

## Controls

[CtlGetSliderValues](#)  
[CtlNewGraphicControl](#)  
[CtlNewSliderControl](#)

[CtlSetGraphics](#)  
[CtlSetSliderValues](#)

These functions add support for graphical buttons and slider controls. For more information, see the section “[Offscreen Windows](#)” on page 91 of the *Palm OS Programmer’s Companion*, vol. I.

## Forms

[FrmCustomResponseAlert](#)      [FrmSetGadgetHandler](#)  
[FrmNewGsi](#)

Among the changes to form functions is extended gadget support. For more information on gadgets and extended gadgets, see the section “[Custom UI Objects \(Gadgets\)](#)” on page 140 of the *Palm OS Programmer’s Companion*, vol. I.

## Menus

[MenuAddItem](#)  
[MenuCmdBarDisplay](#)  
[MenuHideItem](#)

[MenuCmdBarAddButton](#)  
[MenuCmdBarGetButtonData](#)  
[MenuShowItem](#)

For more information on using menu functions, see the section “[Menus](#)” on page 105 of the *Palm OS Programmer’s Companion*, vol. I.

## Overlay Manager

[OmGetCurrentLocale](#)  
[OmGetIndexedLocale](#)  
[OmGetRoutineAddress](#)  
[OmGetSystemLocale](#)

[OmLocaleToOverlayDBName](#)  
[OmOverlayDBNameToLocale](#)  
[OmSetSystemLocale](#)

For more information on using the overlay manager, see the section “[Using Overlays to Localize Resources](#)” on page 365 of the *Palm OS Programmer’s Companion*, vol. I.

## Private Records

[SecSelectViewStatus](#)

[SecVerifyPW](#)

## Compatibility Guide

### 3.5 New Feature Set

---

#### Tables

[TblGetItemPtr](#)  
[TblRowMasked](#)

[TblSetColumnMasked](#)  
[TblSetRowMasked](#)

#### UI Colors

[UIColorGetTableEntryIndex](#)      [UIColorSetTableEntry](#)  
[UIColorGetTableEntryRGB](#)

For more information on using the UI Colors API, see the section “[Color and Grayscale Support](#)” on page 144 of the *Palm OS Programmer’s Companion*, vol. I.

#### UI Controls

[UIBrightnessAdjust](#)

[UIPickColor](#)

#### Windows

[WinCreateBitmapWindow](#)  
[WinDrawPixel](#)  
[WinErasePixel](#)  
[WinGetBitmap](#)  
[WinGetPatternType](#)  
[WinGetPixel](#)  
[WinIndexToRGB](#)  
[WinInvertPixel](#)  
[WinPaintBitmap](#)  
[WinPaintChar](#)  
[WinPaintChars](#)  
[WinPaintLine](#)  
[WinPaintLines](#)  
[WinPaintPixel](#)  
[WinPaintPixels](#)

[WinPaintRectangle](#)  
[WinPaintRectangleFrame](#)  
[WinPalette](#)  
[WinPopDrawState](#)  
[WinPushDrawState](#)  
[WinRGBToIndex](#)  
[WinScreenLock](#)  
[WinScreenMode](#)  
[WinScreenUnlock](#)  
[WinSetBackColor](#)  
[WinSetDrawMode](#)  
[WinSetForeColor](#)  
[WinSetPatternType](#)  
[WinSetTextColor](#)

For more information on using the window functions, see the section “[Drawing on the Palm Powered Handheld](#)” on page 72 of the *Palm OS Programmer’s Companion*, vol. I.

## Miscellaneous New Functions

[DmOpenDBNoOverlay](#)  
[ExgDoDialog](#)  
[DateToAscii](#)

[ResLoadConstant](#)  
[TxtParamString](#)

## Existing Functions that Changed

The following functions that existed prior to 3.5 have changed in release 3.5:

ScrDisplayMode was changed to [WinScreenMode](#).

ContrastAdjust was changed to [UIContrastAdjust](#).

[SelectTime](#) (old function renamed [SelectTimeV33](#))

## New Data Types

The data types Byte, Word, DWord and so on are now deprecated. It is recommend that you use the corresponding new data types. For example, use Int16 instead of SWord and UInt32 instead of DWord. In particular, the unfortunate distinction between Handle/VoidHand has been fixed; use MemHandle instead.

To learn in general how the type names changed, see the header file PalmOSCompatibility.h. This file provides a mapping from the old type name to the new type name. If you need to move forward without modifying your code, you can include this file in your project to provide declarations for the old type names.

## Changes to Events

- The tapCount field has been added to the [EventType](#) structure. The tapCount field specifies the number of times the user tapped the pen at the current location; in fields, two taps selects a word, and three taps selects a line.

## Compatibility Guide

### 3.5 New Feature Set

---

**IMPORTANT:** Because the `tapCount` field has been added to the `EventType` structure, it has become more critical that you clear the event structure before using it to add a new event to the queue. Otherwise, the `tapCount` will be incorrect for the new event.

---

- The structures for `ctlRepeatEvent` and `ctlSelectEvent` have a `value` field added to them. This new field is used only for sliders; it holds the current value of the slider.
- Form objects now handle the `frmTitleSelectEvent` by adding a `keyDownEvent` with the `vchrMenu` character to the event queue (which causes the form's menu to display).
- Some of the structure definitions for system-level events have moved from `Event.h` to `SysEvent.h`.
- The `winEnterEvent` is now not generated until `FrmDrawForm` is called. Make sure to draw your form in response to `frmOpenEvent`, not `winEnterEvent`.
- `EvtSetNullEventTick` is now a function. In previous releases it was a macro.

## Other Changes

- [FrmDrawForm](#)

On release 3.5, `FrmDrawForm` erases the window's rectangle before it draws, so you must perform custom drawing after the call to `FrmDrawForm`, not before. If you have drawn before the call to `FrmDrawForm`, your changes are lost. On debug ROMs, the window handle is invalid until `FrmDrawForm` is called so that draws before `FrmDrawForm` result in a bus error.

- Resource Manager

The resource manager functions have been updated to work with overlay databases. See “[Using Overlays to Localize Resources](#)” on page 365 of the *Palm OS Programmer’s Companion*, vol. I.

- [DmGetDatabase](#)

The order in which this call returns databases has changed. Previously all of the databases from ROM were returned first, then all from RAM. Now they are intermingled. Developers should not rely on the order in which databases are returned from this call.

- [StrToLower](#)

This function is different in 3.5 Latin ROMs. Previously it only changed A through Z. Now it also changes high ASCII characters.

- [Time Manager](#)

If you are using a debug ROM, the string buffer is filled with dateStringLength or longStrLength debugging bytes, depending on the dateFormat parameter. For the routines that return the day-of-week name in addition to the date, the size of the buffers has been expanded, so developers need to check the max lengths defined in `DateTIme.h`.

- The format of the storage heap header has changed, thus any existing saved Simulator card images are invalid and should be tossed.

- [Category Data Structures](#)

The data structure [AppInfoType](#) has been documented.

[CategoryCreateList](#) now has a “hide” function with two new constants; `categoryHideEditCategory`, and `categoryDefaultEditCategoryString`.

- [FtrPtrNew](#)

`FtrPtrNew` now allows allocating chunks larger than 64KB.

- Dynamic heap

The dynamic heap is now sized based on the amount of memory available to the system.

## Compatibility Guide

### Notification Feature Set

---

**Table B.1 Dynamic heap size**

Device RAM Size	Heap Size
$x < 2\text{MB}$	64KB
$2\text{MB} \leq x < 4\text{MB}$	128KB
$x \geq 4\text{MB}$	256KB

## Notification Feature Set

You can check that this feature set is implemented by checking for the existence of the notification manager. You can check by calling [FtrGet](#) as follows:

```
err = FtrGet(sysFtrCreator,  
             sysFtrNumNotifyMgrVersion, &value);
```

If the notification manager is part of the system, the value parameter will be non-zero and the returned error should also be zero (for no error).

### Notification Manager

[SysNotifyBroadcast](#)      [SysNotifyRegister](#)  
[SysNotifyBroadcastDeferred](#)      [SysNotifyUnregister](#)

To learn more about the notification manager, see the section “[Notifications](#)” on page 30 of the *Palm OS Programmer’s Companion*, vol. I.

## 4.0 New Feature Set

You can check that this feature set is implemented by checking that the system version is 4.0 or higher. Use this [FtrGet](#) call:

```
err = FtrGet(sysFtrCreator,  
             sysFtrNumROMVersion, &romVersion);
```

The romVersion parameter should be greater than or equal to 0x04003000, which can be constructed using the [sysMakerROMVersion](#) macro:

`sysMakeROMVersion(4, 0, 0, sysROMStageRelease, 0)`

## Launch Codes

This feature set adds the following launch codes:

[sysAppLaunchCmdAttention](#)  
[sysAppLaunchCmdExgGetData](#)  
[sysAppLaunchCmdExgPreview](#)

## Notifications

This feature set adds the following notifications:

[cncNotifyProfileEvent](#)  
[sysExternalConnectorAttachEvent](#)  
[sysExternalConnectorDetachEvent](#)  
[sysNotifyCardInsertedEvent](#)  
[sysNotifyCardRemovedEvent](#)  
[sysNotifyDBDeletedEvent](#)  
[sysNotifyDeleteProtectedEvent](#)  
[sysNotifyDeviceUnlocked](#)  
[sysNotifyGotUsersAttention](#)  
[sysNotifyHelperEvent](#)  
[sysNotifyLocaleChangedEvent](#)  
[sysNotifyNetLibIFMediaEvent](#)  
[sysNotifyRetryEnqueueKey](#)  
[sysNotifyVolumeMountedEvent](#)  
[sysNotifyVolumeUnmountedEvent](#)

## Functions

This feature set adds the following functions:

### Attention Manager

<a href="#"><u>AttnDoSpecialEffects</u></a>	<a href="#"><u>AttnIndicatorEnable</u></a>
<a href="#"><u>AttnForgetIt</u></a>	<a href="#"><u>AttnIndicatorEnabled</u></a>
<a href="#"><u>AttnGetAttention</u></a>	<a href="#"><u>AttnIterate</u></a>
<a href="#"><u>AttnGetCounts</u></a>	<a href="#"><u>AttnListOpen</u></a>
	<a href="#"><u>AttnUpdate</u></a>

## Compatibility Guide

4.0 New Feature Set

---

### Date and Time Manager

[TimeZoneToAscii](#)  
[TimTimeZoneToUTC](#)  
[TimUTCToTimeZone](#)

### Exchange Manager

<a href="#">ExgGetTargetApplication</a>	<a href="#">ExgControl</a>
<a href="#">ExgGetDefaultApplication</a>	<a href="#">ExgNotifyGoto</a>
<a href="#">ExgGetRegisteredApplications</a>	<a href="#">ExgNotifyPreview</a>
<a href="#">ExgSetDefaultApplication</a>	<a href="#">ExgRequest</a>
	<a href="#">ExgGetRegisteredTypes</a>

For more information on the Exchange Manager, see the chapter [Chapter 1, “Object Exchange,”](#) on page 1 of the *Palm OS Programmer’s Companion*, vol. II, *Communications*.

### Exchange Library

<a href="#">ExgLibAccept</a>	<a href="#">ExgLibOpen</a>
<a href="#">ExgLibClose</a>	<a href="#">ExgLibPut</a>
<a href="#">ExgLibConnect</a>	<a href="#">ExgLibReceive</a>
<a href="#">ExgLibControl</a>	<a href="#">ExgLibRequest</a>
<a href="#">ExgLibDisconnect</a>	<a href="#">ExgLibSend</a>
<a href="#">ExgLibGet</a>	<a href="#">ExgLibSleep</a>
<a href="#">ExgLibHandleEvent</a>	<a href="#">ExgLibWake</a>

For more information on the Exchange Library API, see [Chapter 58, “Exchange Library,”](#) on page 1357 of the *Palm OS Programmer’s Companion*, vol. II, *Communications*.

### Locale Manager

[LmGetLocaleSetting](#)      [LmGetNumLocales](#)  
[LmLocaleToIndex](#)

### Miscellaneous UI

[PhoneNumberLookupCustom](#)

## Notification Manager

[SysNotifyBroadcastFromInterrupt](#)

## PDI Library Functions

<a href="#">PdiDefineReaderDictionary</a>	<a href="#">PdiSetEncoding</a>
<a href="#">ary</a>	<a href="#">PdiWriteBeginObject</a>
<a href="#">PdiDefineResizing</a>	<a href="#">PdiWriteEndObject</a>
<a href="#">PdiEnterObject</a>	<a href="#">PdiWriteParameter</a>
<a href="#">PdiLibClose</a>	<a href="#">PdiWriteParameterStr</a>
<a href="#">PdiLibOpen</a>	<a href="#">PdiWriteProperty</a>
<a href="#">PdiParameterPairTest</a>	<a href="#">PdiWritePropertyValueBinaryValue</a>
<a href="#">PdiReaderDelete</a>	<a href="#">PdiWritePropertyFields</a>
<a href="#">PdiReaderNew</a>	<a href="#">PdiWritePropertyStr</a>
<a href="#">PdiReadParameter</a>	<a href="#">PdiWritePropertyValue</a>
<a href="#">PdiReadProperty</a>	<a href="#">PdiWriterDelete</a>
<a href="#">PdiReadPropertyNameField</a>	<a href="#">PdiWriterNew</a>
<a href="#">PdiReadPropertyName</a>	

For more information, see the chapter [Chapter 3, “Personal Data Interchange,”](#) in *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## Sound Manager Functions

<a href="#">SndInterruptSmfIrregardless</a>	<a href="#">SndPlaySmfIrregardless</a>
<a href="#">SndPlaySmfResourceIrregardless</a>	
<a href="#">S</a>	

For more information, see the chapter [Chapter 10, “Palm System Support,”](#) in *Palm OS Programmer’s Companion*, vol. I.

# Compatibility Guide

## 4.0 New Feature Set

---

### Telephony Manager Functions

<a href="#">TelCancel</a>	<a href="#">TelInfGetInformation</a>
<a href="#">TelCfgGetPhoneNumber</a>	<a href="#">TelIs&lt;FunctionName&gt;Supported</a>
<a href="#">TelCfgGetSmsCenter</a>	<a href="#">TelIs&lt;ServiceSet&gt;Available</a>
<a href="#">TelCfgGetSmsCenter</a>	<a href="#">TelIsCfgServiceAvailable</a>
<a href="#">TelClose</a>	<a href="#">TelIsDtcServiceAvailable</a>
<a href="#">TelClosePhoneConnection</a>	<a href="#">TelIsEmcServiceAvailable</a>
<a href="#">TelDtcCallNumber</a>	<a href="#">TelIsInfServiceAvailable</a>
<a href="#">TelDtcCloseLine</a>	<a href="#">TelIsNwkServiceAvailable</a>
<a href="#">TelDtcReceiveData</a>	<a href="#">TelIsOemServiceAvailable</a>
<a href="#">TelDtcSendData</a>	<a href="#">TelIsPhbServiceAvailable</a>
<a href="#">TelEmcCall</a>	<a href="#">TelIsPhoneConnected</a>
<a href="#">TelEmcCloseLine</a>	<a href="#">TelIsPowServiceAvailable</a>
<a href="#">TelEmcGetNumber</a>	<a href="#">TelIsSmsServiceAvailable</a>
<a href="#">TelEmcGetNumberCount</a>	<a href="#">TelIsSndServiceAvailable</a>
<a href="#">TelEmcSelectNumber</a>	<a href="#">TelIsSpcServiceAvailable</a>
<a href="#">TelEmcSetNumber</a>	<a href="#">TelIsStyServiceAvailable</a>
<a href="#">TelGetCallState</a>	<a href="#">TelMatchPhoneDriver</a>
<a href="#">TelGetEvent</a>	
<a href="#">TelGetTelephonyEvent</a>	
<a href="#">TelNwkGetLocation</a>	<a href="#">TelPhbAddEntry</a>
<a href="#">TelNwkGetNetworkName</a>	<a href="#">TelPhbDeleteEntry</a>
<a href="#">TelNwkGetNetworks</a>	<a href="#">TelPhbGetAvailablePhonebooks</a>
<a href="#">TelNwkGetNetworkType</a>	<a href="#">TelPhbGetEntries</a>
<a href="#">TelNwkGetSearchMode</a>	<a href="#">TelPhbGetEntry</a>
<a href="#">TelNwkGetSelectedNetwork</a>	<a href="#">TelPhbGetEntryCount</a>
<a href="#">TelNwkGetSignalLevel</a>	<a href="#">TelPhbGetEntryMaxSizes</a>
<a href="#">TelNwkSelectNetwork</a>	<a href="#">TelPhbGetSelectedPhonebook</a>
<a href="#">TelNwkSetSearchMode</a>	<a href="#">TelPhbSelectPhonebook</a>
<a href="#">TelOemCall</a>	<a href="#">TelPowGetBatteryStatus</a>
<a href="#">TelOpen</a>	<a href="#">TelPowGetPowerLevel</a>
<a href="#">TelOpenPhoneConnection</a>	<a href="#">TelPowSetPhonePower</a>
	<a href="#">TelSendCommandString</a>

<a href="#">TelSmsDeleteMessage</a>	<a href="#">TelSndPlayKeyTone</a>
<a href="#">TelSmsGetAvailableStorage</a>	<a href="#">TelSndStopKeyTone</a>
<a href="#">e</a>	<a href="#">TelSpcAcceptCall</a>
<a href="#">TelSmsGetDataMaxSize</a>	<a href="#">TelSpcCallNumber</a>
<a href="#">TelSmsGetMessageCount</a>	<a href="#">TelSpcCloseLine</a>
<a href="#">TelSmsGetSelectedStorage</a>	<a href="#">TelSpcConference</a>
<a href="#">TelSmsGetUniquePartId</a>	<a href="#">TelSpcGetCallerNumber</a>
<a href="#">TelSmsReadMessage</a>	<a href="#">TelSpcHoldLine</a>
<a href="#">TelSmsReadMessages</a>	<a href="#">TelSpcPlayDTMF</a>
<a href="#">TelSmsReadReport</a>	<a href="#">TelSpcRejectCall</a>
<a href="#">TelSmsReadReports</a>	<a href="#">TelSpcRetrieveHeldLine</a>
<a href="#">TelSmsReadSubmittedMessage</a>	<a href="#">TelSpcSelectLine</a>
<a href="#">ge</a>	<a href="#">TelSpcSendBurstDTMF</a>
<a href="#">TelSmsReadSubmittedMessages</a>	<a href="#">TelSpcStartContinuousDTMF</a>
<a href="#">TelSmsSelectStorage</a>	<a href="#">TelSpcStopContinuousDTMF</a>
<a href="#">TelSmsSendManualAcknowledgementCode</a>	<a href="#">TelStyChangeAuthenticationCode</a>
<a href="#">dge</a>	<a href="#">TelStyEnterAuthenticationCode</a>
<a href="#">TelSmsSendMessage</a>	<a href="#">TelStyGetAuthenticationState</a>
<a href="#">TelSndMute</a>	

For more information about the Telephony Manager, see [Chapter 10, “Telephony Manager”](#) in *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## Windows Manager

<a href="#">WinGetPixelRGB</a>	<a href="#">WinSetBackColorRGB</a>
<a href="#">WinSetForeColorRGB</a>	<a href="#">WinSetTextRGB</a>

## Internationalization Functions

The following functions have been added to aid with creating localized applications:

<a href="#">FntWCharWidth</a>	<a href="#">OmGetNextSystemLocale</a>
<a href="#">StrCompareAscii</a>	<a href="#">TxtConvertEncoding</a>
<a href="#">TxtGetWordWrapOffset</a>	<a href="#">TxtNameToEncoding</a>
<a href="#">IntlSetRoutineAddress</a>	

## Existing Functions that Changed

The following functions that existed prior to 4.0 have changed in release 4.0:

[ExgRegisterData](#) has been deprecated. The preferred function is now [ExgRegisterDatatype](#).

[ExgGet](#) was previously unimplemented. It has been implemented in release 4.0.

[ExgNotifyReceive](#) was previously a private function. It is now a public function, intended to be used by exchange libraries.

[FldRecalculateField](#) now updates the word-wrapping information whenever FldRecalculateField is called, regardless of the value of the redraw parameter. Prior to Palm OS 4.0 it updated the word-wrapping information only if the redraw parameter was set to true.

[StrNCaselessCompare](#) now requires both of its string parameters to be null-terminated.

[StrNCompare](#) now requires both of its string parameters to be null-terminated.

## Expansion Manager Feature Set

Because not every system has (or needs) Expansion Manager services, applications wishing to use these services should check to make sure they are present before calling them. This is accomplished by checking for the Expansion Manager's system feature with a call to [FtrGet](#), supplying sysFileCExpansionMgr for the feature creator and expFtrIDVersion for the feature number.

The following code shows how to check for the presence and proper version of the Expansion Manager. Note that expectedExpMgrVersionNum should be replaced by the actual version number you expect.

```
UInt32 expMgrVersion;
Err err;

err = FtrGet(sysFileCExpansionMgr,
    expFtrIDVersion, &expMgrVersion);
if(err){
    // Expansion Manager not installed
} else {
    // check version number of Expansion Manager,
    // if necessary
    if(expMgrVersion == expectedExpMgrVersionNum)
        // everything is OK
}
```

---

You can learn more about the expansion manager by reading [Chapter 29, “Expansion Manager”](#) on page 653.

## Functions

This feature set adds the following functions.

### Expansion Manager Functions

<a href="#">ExpCardGetSerialPort</a>	<a href="#">ExpSlotDriverRemove</a>
<a href="#">ExpCardInfo</a>	<a href="#">ExpSlotEnumerate</a>
<a href="#">ExpCardPresent</a>	<a href="#">ExpSlotLibFind</a>
<a href="#">ExpSlotDriverInstall</a>	

## VFS Manager Feature Set

Because not every system has (or needs) Virtual File System (VFS) Manager services, applications wishing to use these services should check to make sure they are present before calling them. This is accomplished by checking for the VFS Manager’s system feature with a call to [FtrGet](#), supplying `sysFileCVFSMgr` for the feature creator and `vfsFtrIDVersion` for the feature number.

The following code shows how to check for the presence and proper version of the VFS Manger. Note that `expectedVFSMgrVersionNum` should be replaced by the actual version number you expect.

## Compatibility Guide

### VFS Manager Feature Set

---

```
UInt32 vfsMgrVersion;
Err err;

err = FtrGet(sysFileCVFSMgr,
    vfsFtrIDVersion, &vfsMgrVersion);
if(err){
    // VFS Manager not installed
} else {
    // check version number of VFS Manager,
    // if necessary
    if(vfsMgrVersion == expectedVFSMgrVersionNum)
        // everything is OK
}
```

---

You can learn more about the VFS manager by reading [Chapter 53, “Virtual File System Manager,”](#) on page 1075.

## Functions

This feature set adds the following functions.

### VFS Manager Functions

<a href="#">VFSCustomControl</a>	<a href="#">VFSFileSize</a>
<a href="#">VFSCustomControl</a>	<a href="#">VFSFileTell</a>
<a href="#">VFSDirEntryEnumerate</a>	<a href="#">VFSFileWrite</a>
<a href="#">VFSExportDatabaseToFile</a>	<a href="#">VFSGetDefaultDirectory</a>
<a href="#">VFSExportDatabaseToFileCust</a>	<a href="#">VFSImportDatabaseFromFile</a>
<a href="#">VFSFileClose</a>	<a href="#">VFSImportDatabaseFromFileCustom</a>
<a href="#">VFSFileCreate</a>	<a href="#">VFSInstallFSLib</a>
<a href="#">VFSFileDBGetRecord</a>	<a href="#">VFSRegisterDefaultDirectory</a>
<a href="#">VFSFileDBGetResource</a>	<a href="#">VFSRemoveFSLib</a>

<a href="#">VFSFileInfo</a>	<a href="#">VFSUnregisterDefaultDirectory</a>
<a href="#">VFSFileDelete</a>	<a href="#">VFSVolumeEnumerate</a>
<a href="#">VFSFileEOF</a>	<a href="#">VFSVolumeFormat</a>
<a href="#">VFSFileGetAttributes</a>	<a href="#">VFSVolumeGetLabel</a>
<a href="#">VFSFileGetDate</a>	<a href="#">VFSVolumeInfo</a>
<a href="#">VFSFileOpen</a>	<a href="#">VFSVolumeMount</a>
<a href="#">VFSFileRead</a>	<a href="#">VFSVolumeSetLabel</a>
<a href="#">VFSFileReadData</a>	<a href="#">VFSVolumeSize</a>
<a href="#">VFSFileRename</a>	<a href="#">VFSVolumeUnmount</a>
<a href="#">VFSFileResize</a>	
<a href="#">VFSFileSetAttributes</a>	
<a href="#">VFSFileSetDate</a>	
<a href="#">VFSFileSeek</a>	

## Bluetooth Library Feature Set

Because not every system has (or needs) Bluetooth Library services, applications wishing to use these services should check to make sure they are present before calling them. This is accomplished by checking for the Bluetooth Library's system feature with a call to [FtrGet](#), supplying `btLibFeatureCreator` for the feature creator and `btLibFeatureVersion` for the feature number.

The following code shows how to check for the presence of the Bluetooth Library.

---

```
UInt32 btVersion;

// Make sure Bluetooth components are installed
// This check also ensures Palm OS 4.0 or greater
if (FtrGet(btLibFeatureCreator, btLibFeatureVersion,
    &btVersion) != errNone) {
    // Alert the user if it's the active application
```

```
if ((launchFlags & sysAppLaunchFlagNewGlobals) &&
    (launchFlags & sysAppLaunchFlagUIApp))
    FrmAlert (MissingBtComponentsAlert);
    return sysErrRomIncompatible;
}
```

---

You can learn more about the Bluetooth Library by reading [Chapter 6, “Bluetooth,”](#) on page 131 of the *Palm OS Programmer’s Companion*, vol. II, *Communications*.

## Functions

This feature set adds the following functions.

### Bluetooth Library Security Functions

[BtLibSecurityFindTrustedDeviceRecord](#) [BtLibSecurityNumTrustedDeviceRecords](#)

[BtLibSecurityGetTrustedDeviceRecordInfo](#) [BtLibSecurityRemoveTrustedDevice Record](#)

### Bluetooth Library Utility Functions and Macros

[BtLibAddrAToBtd](#) [BtLibRfCommHToNS](#)

[BtLibAddrBtdToA](#) [BtLibRfCommNToHL](#)

[BtLibL2CapHToNL](#) [BtLibRfCommNToHS](#)

[BtLibL2CapHToNS](#) [BtLibSdpHToNL](#)

[BtLibL2CapNToHL](#) [BtLibSdpHToNS](#)

[BtLibL2CapNToHS](#) [BtLibSdpNToHL](#)

[BtLibRfCommHToNL](#) [BtLibSdpNToHS](#)

### Bluetooth Library Management Functions

[BtLibClose](#) [BtLibLinkSetState](#)

[BtLibOpen](#) [BtLibPiconetCreate](#)

[BtLibCancelInquiry](#) [BtLibPiconetDestroy](#)

<a href="#">BtLibDiscoverMultipleDevices</a>	<a href="#">BtLibPiconetLockInbound</a>
<a href="#">BtLibDiscoverSingleDevice</a>	<a href="#">BtLibPiconetUnlockInbound</a>
<a href="#">BtLibGetGeneralPreference</a>	<a href="#">BtLibRegisterManagementNotification</a>
<a href="#">BtLibGetRemoteDeviceName</a>	<a href="#">BtLibSetGeneralPreference</a>
<a href="#">BtLibGetSelectedDevices</a>	<a href="#">BtLibStartInquiry</a>
<a href="#">BtLibLinkConnect</a>	<a href="#">BtLibUnregisterManagementNotification</a>
<a href="#">BtLibLinkDisconnect</a>	<a href="#">BtLibManagementCallback</a>
<a href="#">BtLibLinkGetState</a>	

### **Bluetooth Library Socket Functions**

<a href="#">BtLibSocketAdvanceCredit</a>	<a href="#">BtLibSocketGetInfo</a>
<a href="#">BtLibSocketClose</a>	<a href="#">BtLibSocketListen</a>
<a href="#">BtLibSocketConnect</a>	<a href="#">BtLibSocketRespondToConnection</a>
<a href="#">BtLibSocketCreate</a>	<a href="#">BtLibSocketSend</a>

### **Bluetooth Library Service Discovery Protocol Functions**

<a href="#">BtLibSdpCompareUuids</a>	<a href="#">BtLibSdpServiceRecordGetSizeOfRawAttribute</a>
<a href="#">BtLibSdpGetPSMByUuid</a>	<a href="#">BtLibSdpServiceRecordGetStringOrUrlLength</a>
<a href="#">BtLibSdpGetRawDataElementSize</a>	<a href="#">BtLibSdpServiceRecordMapRemote</a>
<a href="#">BtLibSdpGetRawDataElementType</a>	<a href="#">BtLibSdpServiceRecordSetAttribute</a>

## Compatibility Guide

### High-Density Display Feature Set

---

[BtLibSdpGetServerChannel](#) [BtLibSdpServiceRecordSet](#)  
[ByUuid](#) [AttributesForSocket](#)

[BtLibSdpParseRawDataElem](#) [BtLibSdpServiceRecordSet](#)  
[ent](#) [RawAttribute](#)

[BtLibSdpServiceRecordCre](#) [BtLibSdpServiceRecordsGe](#)  
[ate](#) [tByServiceClass](#)

[BtLibSdpServiceRecordDes](#) [BtLibSdpServiceRecordSta](#)  
[troy](#) [rtAdvertising](#)

[BtLibSdpServiceRecordGet](#) [BtLibSdpServiceRecordSto](#)  
[Attribute](#) [pAdvertising](#)

[BtLibSdpServiceRecordGet](#) [BtLibSdpUuidInitialize](#)  
[NumListEntries](#)

[BtLibSdpServiceRecordGet](#) [BtLibSdpVerifyRawDataEle](#)  
[NumLists](#) [ment](#)

[BtLibSdpServiceRecordGet](#) [BtLibSocketCallback](#)  
[RawAttribute](#)

## High-Density Display Feature Set

You can verify that this feature set is implemented by checking the version of the Window Manager. If the Window Manager version is 4 or greater, the High-Density Display feature set is supported. To check the version of the Window Manager, use this call:

```
err = FtrGet(sysFtrCreator,  
             sysFtrNumWinVersion, &version);
```

Upon return, if `version` has a value of 4 or greater, the High-Density Display feature set is present. Note that just because the High-Density Display feature set is present, it isn't necessarily being used. You may want to check the density of the screen, as follows:

---

```
WinScreenGetAttribute(winScreenDensity, &attr);  
if (attr == kDensityDouble) {  
    //the screen is high density  
}
```

---

The [5.0 New Feature Set](#) incorporates all of the functionality present in the High-Density Display Feature Set, so if your application is running on Palm OS 5 you can assume that the High-Density Display Feature Set is present as well.

## New Data Types

This feature set adds the following new data types.

### Bitmap Data Types

<a href="#">BitmapTypeV0</a>	<a href="#">BitmapTypeV1</a>
<a href="#">BitmapTypeV2</a>	<a href="#">BitmapTypeV3</a>
<a href="#">DensityType</a>	<a href="#">PixelFormatType</a>

In addition, the definition of [BitmapType](#) changed.

### Font Data Types

<a href="#">FontDensityType</a>	<a href="#">FontTypeV2</a>
---------------------------------	----------------------------

In addition, this feature set defines a new [Extended Font Resource](#).

### Window Constants

This feature set adds a new set of [Window Coordinate System Constants](#).

## Functions

This feature set adds the following functions.

### Bitmap Functions

<a href="#">BmpCreateBitmapV3</a>	<a href="#">BmpGetCompressionType</a>
<a href="#">BmpGetDensity</a>	<a href="#">BmpGetNextBitmapAnyDensity</a>
<a href="#">BmpGetTransparentValue</a>	<a href="#">BmpGetVersion</a>
<a href="#">BmpSetDensity</a>	<a href="#">BmpSetTransparentValue</a>

## Compatibility Guide

### Sound Stream Feature Set

---

#### System Event Manager Functions

[EvtGetPenNative](#)

#### Window Functions

[WinGetCoordinateSystem](#)    [WinGetSupportedDensity](#)

[WinPaintRoundedRectangle](#) [WinPaintTiledBitmapFrame](#)

[WinScaleCoord](#)

[WinScalePoint](#)

[WinScaleRectangle](#)

[WinScreenGetAttribute](#)

[WinSetCoordinateSystem](#)

[WinUnscaleCoord](#)

[WinUnscalePoint](#)

[WinUnscaleRectangle](#)

## Sound Stream Feature Set

The Sound Stream feature set adds a number of “stream” functions and constants to the Sound Manager. You can verify that this feature set is supported by checking for the Sound Manager’s version number. If the Sound Manager’s version feature is defined, the Sound Stream feature set is supported.

The following code shows how to check for the presence and proper version of the Sound Manager. Note that `expectedSndMgrVersionNum` should be replaced by the actual version number you expect (typically, `sndMgrVersionNum`).

---

```
UInt32 version;
Err err;

err = FtrGet(sysFileCSoundMgr, sndFtrIDVersion, &version);
if(err) {
    // Sound Stream Feature Set not present
} else {
    // The Sound Stream Feature Set is present.
    // Check version number of Sound Manager,
    // if necessary
    if(version == expectedSndMgrVersionNum)
        // everything is OK
}
```

---

## **Sound Stream Data Structures and Types**

This feature set adds the following data structures:

[SndPtr](#)

[SndStreamRef](#)

[SndSampleType](#)

[SndStreamWidth](#)

[SndStreamMode](#)

## **Sound Stream Enums and Constants**

This feature set adds the following enums and constants:

### **Sound Stream Enums and Constants**

[SndSampleTypeTag](#)

[Stereo Pan Constants](#)

[SndStreamModeTag](#)

[Volume Constants](#)

[SndStreamWidthTag](#)

[Sound Resource Playback Flags](#)

## **Sound Stream Functions**

This feature set adds the following functions.

### **Sound Stream Functions**

[SndPlayResource](#)

[SndStreamPause](#)

[SndStreamCreate](#)

[SndStreamSetPan](#)

[SndStreamDelete](#)

[SndStreamSetVolume](#)

[SndStreamGetPan](#)

[SndStreamStart](#)

[SndStreamGetVolume](#)

[SndStreamStop](#)

In addition, the Sound Stream Feature Set defines the following callback function:

[SndStreamBufferCallback](#)

## 5.0 New Feature Set

You can check that this feature set is implemented by checking that the system version is 5.0 or higher. Use this [FtrGet](#) call:

```
err = FtrGet(sysFtrCreator,  
             sysFtrNumROMVersion, &romVersion);
```

The romVersion parameter should be greater than or equal to 0x05003000, which can be constructed using the [sysMakeROMVersion](#) macro:

```
sysMakeROMVersion(5, 0, 0, sysROMStageRelease, 0)
```

This feature set corresponds to version 5.0 of Palm OS 5.

The Palm OS Application Compatibility Environment (PACE) has its own version associated with it. You can obtain this version number with:

```
err = FtrGet('pace', 0, &paceVersion);
```

## Notifications

This feature set adds the following database-related notifications:

[sysNotifyDBCreatedEvent](#)   [sysNotifyDBDirtyEvent](#)  
[sysNotifyDBChangedEvent](#)

It also broadcasts the following notifications:

[sysNotifyAppLaunchingEvent](#)  
[sysNotifyAppQuittingEvent](#)  
[sysNotifyEventDequeuedEvent](#)  
[sysNotifyIdleTimeEvent](#)  
[sysNotifyInsPtEnableEvent](#)  
[sysNotifyKeyboardDialogEvent](#)  
[sysNotifyProcessPenStrokeEvent](#)  
[sysNotifyVirtualCharHandlingEvent](#)

## Functions

This feature set adds the following functions.

## ARM-Native Functions

[PceNativeCall](#)

## Functions and Traps not Supported by PACE

For various reasons a number of functions and traps are not supported by PACE. The following sections list those functions, grouped according to the reason that they are not implemented.

### Unimplemented “System Use Only” Functions

The following functions, which are documented as “System Use Only,” are not supported by the Palm OS Application Compatibility Environment (PACE).

## Compatibility Guide

### 5.0 New Feature Set

---

AlmAlarmCallback	MemPtrResetLock
AlmCancelAll	MemStoreSetInfo
AlmDisplayAlarm	PenClose
AlmInit	PenGetRawPen
AlmTimeChange	PenOpen
DmInit	ScrCompressScanLine
EvtDequeueKeyEvent	ScrCopyRectangle
EvtGetSysEvent <sup>1</sup>	ScrDecompressScanLine
EvtInitialize	ScrDrawChars
EvtSetKeyQueuePtr	ScrDrawNotify
EvtSetPenQueuePtr	ScrLineRoutine
EvtSysInit	ScrRectangleRoutine
ExgInit	ScrScreenInfo
FrmAddSpaceForObject	ScrSendUpdateArea
FtrInit	SlkProcessRPC
GrfFree	SlkSysPktDefaultResponse
GrfInit	SndInit
InsPtCheckBlink	SysBatteryDialog
InsPtInitialize	SysColdBoot
IntlInit	SysDoze
MemCardFormat	SysInit
MemHandleFlags	SysNewOwnerID
MemHandleOwner	SysSemaphoreSet
MemHandleResetLock <sup>1</sup>	SysUILaunch
MemHeapFreeByOwnerId	SysWantEvent
MemHeapInit	TimInit
MemInit	UIInitialize
MemInitHeapTable	UIReset
MemKernelInit	WinAddWindow
MemPtrFlags	WinRemoveWindow
MemPtrOwner	

1. Implemented on release ROMs but flagged as illegal on debug ROMs.

As a rule, functions and traps that are not documented should be treated as if they are unimplemented and should not be used by applications.

## Implemented “System Use Only” Functions and Traps

Although marked “System Use Only,” a number of functions and traps are required by applications in the ROM, by Palm Debugger, by test applications, by scripting, or by some popular applications. Because of this, the following System Use Only functions and traps are supported by PACE. Because they are intended only for system use, however, applications should do what they can to avoid using them.

AlmEnableNotification	HwrDockStatus
AttnAllowClose	HwrLEDAttributes
AttnEnableNotification	HwrMemReadable
AttnIndicatorAllow	HwrMemWritable
AttnIndicatorAllowed	HwrVibrateAttributes
AttnIndicatorGetBlinkPat	MemChunkNew
tern	MemHeapPtr
AttnIndicatorSetBlinkPat	PenRawToScreen
tern	PenScreenToRaw
AttnReopen	SysGetAppInfo
DmResetRecordStates	SysLaunchConsole
EvtEnqueuePenPoint	SndSetDefaultVolume
<a href="#"><u>EvtGetSilkscreenAreaList</u></a>	SysLCDContrast
FileReadLow	SysSetA5
Find	SysSetPerformance
FrmActiveState	SysSleep
<a href="#"><u>FrmHandleEvent</u></a>	SysUIBusy
HwrDelay	WinDrawWindowFrame

## Obsolete Functions and Traps

The following functions are not supported by PACE because they are obsolete.

<a href="#"><u>FplAdd</u></a> <sup>1</sup>	<a href="#"><u>FplFToA</u></a> <sup>1</sup>
<a href="#"><u>FplAToF</u></a> <sup>1</sup>	<a href="#"><u>FplLongToFloat</u></a> <sup>1</sup>
<a href="#"><u>FplBase10Info</u></a> <sup>1</sup>	<a href="#"><u>FplMul</u></a> <sup>1</sup>
<a href="#"><u>FplDiv</u></a> <sup>1</sup>	<a href="#"><u>FplSub</u></a> <sup>1</sup>
<a href="#"><u>FplFloatToLong</u></a> <sup>1</sup>	WiCmdV32
<a href="#"><u>FplFloatToULong</u></a> <sup>1</sup>	

1. Implemented on release ROMs but flagged as illegal on debug ROMs.

## Compatibility Guide

### 5.0 New Feature Set

---

#### 'NOP' Functions and Traps

These functions and traps should not be called by applications (many are documented as "System Use Only"). Because some third-party applications do call them, for backward compatibility they act as NOPs.

<a href="#"><u>FplFree</u></a>	SerReceiveISP
<a href="#"><u>FplInit</u></a>	SrmSleep
HwrEnableDataWrites	SrmWake
HwrDisableDataWrites	SysDisableInts
HwrTimerSleep	SysRestoreStatus
HwrTimerWake	TimHandleInterrupt
KeyResetDoubleTap	TimSleep
KeySleep	TimWake
KeyWake	WinDisableWindow
PenSleep	WinEnableWindow
PenWake	WinInitializeWindow

#### Unimplemented Rare Functions and Traps

These functions and traps are only used internally by Palm OS, by serial drivers, by OEM extensions, and the like. They are not implemented by PACE.

ConGetS	FlashErase
ConPutS	FlashProgram
<a href="#"><u>DayDrawDays</u></a>	<a href="#"><u>IntlSetRoutineAddress</u></a>
<a href="#"><u>DayDrawDaySelector</u></a>	MemGetRomNVParams
DbgCommSettings	MemNVParams
DbgGetMessage	OEMDispatch <sup>1</sup>
DlkDispatchRequest	<a href="#"><u>ResLoadForm</u></a>
DlkStartServer	<a href="#"><u>SlkSetSocketListener</u></a>
DmMoveOpenDBContext	SysNotifyDatabaseAdded
DmOpenDBWithLocale	SysNotifyDatabaseRemoved
FlashCompress	<a href="#"><u>SysSetTrapAddress</u></a>

1. Supported if the OEM supports the trap.

## 5.1 New Feature Set

You can check that this feature set is implemented by checking that the system version is 5.1 or higher. Use this [FtrGet](#) call:

```
err = FtrGet(sysFtrCreator,  
    sysFtrNumROMVersion, &romVersion);
```

The romVersion parameter should be greater than or equal to 0x05103000, which can be constructed using the [sysMakeROMVersion](#) macro:

```
sysMakeROMVersion(5, 1, 0, sysROMStageRelease, 0)
```

This feature set corresponds to version 5.1 of Palm OS 5.

### Net Library Interface Settings

This feature set adds the following Net Library Interface Settings:

DriverVersion	80211AccessPointBSSID
FirmwareVersion	80211AssociationStatus
FirmwareDate	80211MKKCallSign
80211Device	80211CountryTest
80211ESSID	

See [NetLibIFSettingGet](#) for a description of each of the above settings.

### CPM Library

This feature set includes the Cryptographic Provider Manager (CPM) shared library. The CPM library isn't automatically loaded upon system boot: before making use of the CPM library you must first load it, using [SysLibFind](#) and [SysLibLoad](#).

The CPM library defines a number of constants:

- [AP Capability Constants](#)
- [Block Encryption Mode Constants](#)
- [Cipher Algorithm Constants](#)
- [Export Encoding Constants](#)
- [Hashing Algorithm Constants](#)

## Compatibility Guide

### 5.1 New Feature Set

---

- [Key Class Constants](#)
- [Key Usage Constants](#)
- [Plaintext Padding Constants](#)

The following structures are defined as part of the CPM library:

<a href="#">APCipherInfoStruct</a>	<a href="#">APPProviderInfoStruct</a>
<a href="#">APHashInfoStruct</a>	<a href="#">APVerifyInfoStruct</a>
<a href="#">APKeyInfoStruct</a>	<a href="#">CPMInfoStruct</a>
<a href="#">APPProviderContextStruct</a>	

Finally, the CPM library contains the following functions:

<a href="#">CPMLibDecrypt</a>	<a href="#">CPMLibHash</a>
<a href="#">CPMLibDecryptFinal</a>	<a href="#">CPMLibHashFinal</a>
<a href="#">CPMLibDecryptInit</a>	<a href="#">CPMLibHashInit</a>
<a href="#">CPMLibDecryptUpdate</a>	<a href="#">CPMLibImportCipherInfo</a>
<a href="#">CPMLibEncrypt</a>	<a href="#">CPMLibImportHashInfo</a>
<a href="#">CPMLibEncryptFinal</a>	<a href="#">CPMLibImportKeyInfo</a>
<a href="#">CPMLibEncryptInit</a>	<a href="#">CPMLibImportVerifyInfo</a>
<a href="#">CPMLibEncryptUpdate</a>	<a href="#">CPMLibReleaseCipherInfo</a>
<a href="#">CPMLibExportCipherInfo</a>	<a href="#">CPMLibReleaseHashInfo</a>
<a href="#">CPMLibExportHashInfo</a>	<a href="#">CPMLibReleaseKeyInfo</a>
<a href="#">CPMLibExportKeyInfo</a>	<a href="#">CPMLibReleaseVerifyInfo</a>
<a href="#">CPMLibExportVerifyInfo</a>	<a href="#">CPMLibVerify</a>
<a href="#">CPMLibGenerateKey</a>	<a href="#">CPMLibVerifyFinal</a>
<a href="#">CPMLibGetInfo</a>	<a href="#">CPMLibVerifyInit</a>
<a href="#">CPMLibGetProviderInfo</a>	<a href="#">CPMLibVerifyUpdate</a>

## SSL Library

The 5.1 New Feature Set includes the Secure Sockets Layer (SSL) shared library. The SSL library isn't automatically loaded upon system boot: before making use of the SSL library you must first load it and open it, using code similar to the following:

---

```
Err error;
UInt16 libRef;

if ( SyLibFind( kSslDBName, &libRef ) != 0 )
{
    error = SysLibLoad(kSslLibType, kSslLibCreator, &libRef);
}
```

```
/* error checking goes here. */  
  
error = SslLibOpen( libRef );  
...
```

---

## SSL Library Structures and Data Types

The SSL library defines a number of structures and data types:

<a href="#">SslAttribute</a>	<a href="#">SslExtendedItems</a>
<a href="#">SslContext</a>	<a href="#">SslIoBuf</a>
<a href="#">SslLib</a>	<a href="#">SslLibCallback</a>
<a href="#">SslCallback</a>	<a href="#">SslSession</a>
<a href="#">SslCipherSuiteInfo</a>	<a href="#">SslSocket</a>
<a href="#">SslExtendedItem</a>	<a href="#">SslVerify</a>

## SSL Library Functions

<a href="#">SslClose</a>	<a href="#">SslLibOpen</a>
<a href="#">SslConsume</a>	<a href="#">SslLibGetLong</a>
<a href="#">SslContextCreate</a>	<a href="#">SslLibGetPtr</a>
<a href="#">SslContextDestroy</a>	<a href="#">SslLibSetLong</a>
<a href="#">SslContextGetLong</a>	<a href="#">SslLibSetPtr</a>
<a href="#">SslContextGetPtr</a>	<a href="#">SslOpen</a>
<a href="#">SslContextSetLong</a>	<a href="#">SslPeek</a>
<a href="#">SslContextSetPtr</a>	<a href="#">SslRead</a>
<a href="#">SslFlush</a>	<a href="#">SslReceive</a>
<a href="#">SslLibClose</a>	<a href="#">SslSend</a>
<a href="#">SslLibCreate</a>	<a href="#">SslWrite</a>
<a href="#">SslLibDestroy</a>	

In addition, the SSL library defines one callback function:

- [SslCallbackFunc](#)

## SSL Library Attributes and Macros

You interact with SSL attributes using the following macros:

- [SslContextGet\\_Attribute \(integer version\)](#)
- [SslContextGet\\_Attribute \(pointer version\)](#)
- [SslLibSet\\_Attribute \(integer version\)](#)
- [SslContextSet\\_Attribute \(pointer version\)](#)

## Compatibility Guide

### 5.1 New Feature Set

---

- [SslLibGet\\_Attribute \(integer version\)](#)
- [SslLibGet\\_Attribute \(pointer version\)](#)
- [SslLibSet\\_Attribute \(integer version\)](#)
- [SslLibSet\\_Attribute \(pointer version\)](#)

The attributes that you can manipulate with the above macros are:

<a href="#"><u>AppInt32</u></a>	<a href="#"><u>LastIO</u></a>
<a href="#"><u>AppPtr</u></a>	<a href="#"><u>Mode</u></a>
<a href="#"><u>AutoFlush</u></a>	<a href="#"><u>PeerCert</u></a>
<a href="#"><u>BufferedReuse</u></a>	<a href="#"><u>PeerCommonName</u></a>
<a href="#"><u>CipherSuite</u></a>	<a href="#"><u>ProtocolVersion</u></a>
<a href="#"><u>CipherSuiteInfo</u></a>	<a href="#"><u>RbufSize</u></a>
<a href="#"><u>CipherSuites</u></a>	<a href="#"><u>ReadBufPending</u></a>
<a href="#"><u>Compat</u></a>	<a href="#"><u>ReadOutstanding</u></a>
<a href="#"><u>DontSendShutdown</u></a>	<a href="#"><u>ReadRecPending</u></a>
<a href="#"><u>DontWaitForShutdown</u></a>	<a href="#"><u>ReadStreaming</u></a>
<a href="#"><u>Error</u></a>	<a href="#"><u>SessionReused</u></a>
<a href="#"><u>HsState</u></a>	<a href="#"><u>Socket</u></a>
<a href="#"><u>InfoCallback</u></a>	<a href="#"><u>SslSession</u></a>
<a href="#"><u>InfoInterest</u></a>	<a href="#"><u>SslVerify</u></a>
<a href="#"><u>IoTimeout</u></a>	<a href="#"><u>Streaming</u></a>
<a href="#"><u>IoFlags</u></a>	<a href="#"><u>VerifyCallback</u></a>
<a href="#"><u>IoStruct</u></a>	<a href="#"><u>WbufSize</u></a>
<a href="#"><u>LastAlert</u></a>	<a href="#"><u>WriteBufPending</u></a>
<a href="#"><u>LastApi</u></a>	

# C

## 1.0 Float Manager

---

This section provides reference material for the Palm OS 1.0 float manager. In Palm OS 1.0, the float manager API is declared in the header file `FloatMgr.h`. In Palm OS 2.0, this file was renamed to `FloatMgrOld.h`. In Palm OS 3.5, this file was made private.

---

**NOTE:** New applications should no longer use the 1.0 Float Manager. See [Chapter 32, “Float Manager,”](#) on page 695 for the functions now provided by the Float Manager.

---

### Float Manager Functions

#### FplAdd

**Purpose** Add two floating-point numbers (returns  $a + b$ ).

**Prototype** `FloatType FplAdd (FloatType a, FloatType b)`

**Parameters** `a, b` The floating-point numbers.

**Result** Returns the normalized floating-point result of the addition.

**Comments** Under Palm OS® 2.0 and later, most applications will want to use the arithmetic symbols instead. See the [“Floating-Point”](#) section in the *Palm OS Programmer’s Companion*, vol. I.

**Compatibility** This function is not supported by PACE ([5.0 New Feature Set](#)).

## 1.0 Float Manager

### *Float Manager Functions*

---

## FplAToF

<b>Purpose</b>	Convert a zero-terminated ASCII string to a floating-point number. The string must be in the format: [-]x[.]yyyyyyyy[e[-]zz]
<b>Prototype</b>	FloatType FplAToF (char* s)
<b>Parameters</b>	s                            Pointer to the ASCII string.
<b>Result</b>	Returns the floating-point number.
<b>Comments</b>	The mantissa of the number is limited to 32 bits.
<b>Compatibility</b>	This function is not supported by PACE ( <a href="#">5.0 New Feature Set</a> ).
<b>See Also</b>	<a href="#">FplFToA</a> , <a href="#">FplFree</a> , <a href="#">FplInit</a>

## FplBase10Info

<b>Purpose</b>	Extract detailed information on the base 10 form of a floating-point number: the base 10 mantissa, exponent, and sign.
<b>Prototype</b>	Err FplBase10Info (FloatType a, ULONG* mantissaP, Int* exponentP, Int* signP)
<b>Parameters</b>	a                            The floating-point number. mantissaP                    The base 10 mantissa (return value). exponentP                    The base 10 exponent (return value). signP                        The sign, 1 or -1 (return value).
<b>Result</b>	Returns an error code, or 0 if no error.
<b>Comments</b>	The mantissa is normalized so it contains at least kMaxSignificantDigits significant digits when printed as an integer value.

`FlpBase10Info` reports that zero is "negative"; that is, it returns a one for `xSign`. If this is a problem, a simple workaround is:

```
if (xMantissa == 0) {  
    xSign = 0;
```

**Compatibility** This function is not supported by PACE ([5.0 New Feature Set](#)).

## FplDiv

**Purpose** Divide two floating-point numbers (result = dividend/divisor).

**Prototype** `FloatType FplDiv (FloatType dividend,  
 FloatType divisor)`

**Parameters** `dividend` Floating-point dividend.  
`divisor` Floating-point divisor.

**Result** Returns the normalized floating-point result of the division.

Under Palm OS 2.0 and later, most applications will want to use the arithmetic symbols instead. See the "[Floating-Point](#)" section in the *Palm OS Programmer's Companion*, vol. I.

**Compatibility** This function is not supported by PACE ([5.0 New Feature Set](#)).

## FplFloatToLong

**Purpose** Convert a floating-point number to a long integer.

**Prototype** `Long FplFloatToLong (FloatType f)`

**Parameters** `f` Floating-point number to be converted.

**Result** Returns the long integer.

## 1.0 Float Manager

### *Float Manager Functions*

---

**Compatibility** This function is not supported by PACE ([5.0 New Feature Set](#)).

**See Also** [FplLongToFloat](#), [FplFloatToULong](#)

## FplFloatToULong

**Purpose** Convert a floating-point number to an unsigned long integer.

**Prototype** `ULong FplFloatToULong (FloatType f)`

**Parameters** `f` Floating-point number to be converted.

**Result** Returns an unsigned long integer.

**Compatibility** This function is not supported by PACE ([5.0 New Feature Set](#)).

**See Also** [FplLongToFloat](#), [FplFloatToLong](#)

## FplFree

**Purpose** Release all memory allocated by the floating-point initialization.

**Prototype** `void FplFree ()`

**Parameters** None.

**Result** Returns nothing.

**Comments** Applications must call this routine after they've called other functions that are part of the float manager.

**Compatibility** If [5.0 New Feature Set](#) is present, this function acts as a NOP.

**See Also** [FplInit](#)

## FplFToA

<b>Purpose</b>	Convert a floating-point number to a zero-terminated ASCII string in exponential format: [-] x.yyyyyyye [-] zz
<b>Prototype</b>	Err FplFToA (FloatType a, char* s)
<b>Parameters</b>	a Floating-point number. s Pointer to buffer to contain the ASCII string.
<b>Result</b>	Returns an error code, or 0 if no error.
<b>Compatibility</b>	This function is not supported by PACE ( <a href="#">5.0 New Feature Set</a> ).
<b>See Also</b>	<a href="#">FplAToF</a> , <a href="#">FplFree</a> , <a href="#">FplInit</a>

## FplInit

<b>Purpose</b>	Initialize the floating-point conversion routines. Allocate space in the system heap for floating-point globals. Initialize the tenPowers array in the globals area to the powers of 10 from -99 to +99 in floating-point format.
<b>Prototype</b>	Err FplInit()
<b>Parameters</b>	None.
<b>Result</b>	Returns an error code, or 0 if no error.
<b>Comments</b>	Applications must call this routine before calling any other Fpl function.
<b>Compatibility</b>	If <a href="#">5.0 New Feature Set</a> is present, this function acts as a NOP.
<b>See Also</b>	<a href="#">FplFree</a>

## 1.0 Float Manager

### *Float Manager Functions*

---

## FplLongToFloat

**Purpose** Convert a long integer to a floating-point number.

**Prototype** `FloatType FplLongToFloat (Long x)`

**Parameters** `x` A long integer.

**Result** Returns the floating-point number.

**Compatibility** This function is not supported by PACE ([5.0 New Feature Set](#)).

## FplMul

**Purpose** Multiply two floating-point numbers.

**Prototype** `FloatType FplMul (FloatType a, FloatType b)`

**Parameters** `a, b` The floating-point numbers.

**Result** Returns the normalized floating-point result of the multiplication.

**Comments** Under Palm OS 2.0 and later, most applications will want to use the arithmetic symbols instead. See the “[Floating-Point](#)” section in the *Palm OS Programmer’s Companion*, vol. I.

**Compatibility** This function is not supported by PACE ([5.0 New Feature Set](#)).

## FplSub

**Purpose** Subtract two floating-point numbers (returns  $a - b$ ).

**Prototype** `FloatType FplSub (FloatType a, FloatType b)`

**Parameters** `a, b` The floating-point numbers.

**Result** Returns the normalized floating-point result of the subtraction.

**Comments** Under Palm OS 2.0 and later, most applications will want to use the arithmetic symbols instead. See the “[Floating-Point](#)” section in the *Palm OS Programmer’s Companion*, vol. I.

**Compatibility** This function is not supported by PACE ([5.0 New Feature Set](#)).

## **1.0 Float Manager**

### *Float Manager Functions*

---

# Index

---

## Symbols

\_searchF 967

## Numerics

2.0 feature set 2304  
3.0 feature set 2308  
3.1 feature set 2312  
3.2 feature set 2315  
3.5 feature set 2324, 2330, 2342, 2346, 2351

## A

accented characters and StrToLower 936  
active form 284, 285  
active window 68, 1170, 1230  
adding event to event queue 942  
AddrLookupParamsType 494  
AlarmMgr.h 505  
alarms 505–509  
    and launch codes 17  
    canceling 507  
    procedure alarms 508  
    setting 507  
    sysAppLaunchCmdTimeChange 35  
alerts 275  
    custom alert 278, 279  
    SysFatalAlert 418  
allocating chunks on dynamic heap 794  
AlmAlarmProcPtr 509  
almErrFull 507, 508  
almErrMemory 507, 508  
AlmGetAlarm 505  
AlmGetProcAlarm 506  
almProcCmdCustom 510  
AlmProcCmdEnum 509  
almProcCmdReschedule 510  
almProcCmdTriggered 510  
AlmSetAlarm 507  
AlmSetProcAlarm 508  
alphaGraffitiSilkscreenArea 954  
APAlgorithmEnum 2085  
APKeyClassEnum 2089  
APKeyUsageEnum 2089  
APModeEnum 2084

APPaddingEnum 2090  
appErrorClass 645  
appEvtHookKeyMask 57  
AppInfoPtr 133  
appInfoStringsRsc 146  
AppInt32 SSL Attribute 2187  
AppLaunchCmd.h 493  
application preferences 842  
applications  
    Security 34  
AppPtr SSL Attribute 2188  
APProviderContextStruct 2095  
APProviderInfoStruct 2095  
APProviderInfoType 2095  
appStopEvent 43  
archiving  
    marking record as archived 571  
ARM-native code  
    calling 1258  
atoi function substitute (StrToInt) 919  
Attention Manager 91, 105  
AttnCallbackProc 130  
AttnCommandArgsType 107  
AttnCommandType 105  
AttnDoSpecialEffects 119  
attnErrMemory 116  
AttnFlagsType 111  
AttnForgetIt 120  
AttnGetAttention 121  
AttnGetCounts 124  
AttnIndicatorEnable 125  
AttnIndicatorEnabled 126  
AttnIterate 126  
AttnLaunchCodeArgsType 113  
AttnLevelType 114  
AttnListOpen 127  
AttnNotifyDetailsType 91  
AttnUpdate 128  
AutoFlush SSL Attribute 2188  
auto-off  
    setting 990  
    timer 957  
autoRepeatKeyMask 57

---

**B**

badDrawWindowValue 1236  
BarBeamBitmap 375  
BarCopyBitmap 374  
BarCutBitmap 374  
BarDeleteBitmap 374  
BarInfoBitmap 375  
BarPasteBitmap 374  
BarSecureBitmap 374  
BarUndoBitmap 374  
base 10 form of floating-point number 698, 2356  
battery timeout 963, 965  
battery voltage warning threshold 963, 965  
Bitmap.h 513  
BitmapCompressionType 513  
BitmapDirectInfoType 514  
BitmapFlagsType 515  
BitmapPtr 518  
bitmapRsc 535  
bitmaps  
  drawing 1172  
BitmapType 518  
BitmapTypeV0 521  
BitmapTypeV1 522  
BitmapTypeV2 524  
BitmapTypeV3 527  
BitmapVersionOne 535  
BitmapVersionTwo 535  
BitmapVersionZero 535  
blank lines in field 213  
BmpBitsSize 536  
BmpColortableSize 537  
BmpCompress 537  
BmpCreate 538  
BmpCreateBitmapV3 541  
BmpDelete 542  
BmpGetBits 543  
BmpGetColortable 544  
BmpGetCompressionType 544  
BmpGetDensity 545  
BmpGetNextBitmapAnyDensity 548  
BmpGetTransparentValue 550  
BmpGetVersion 551

BmpGlueGetBitDepth 544, 1892  
BmpGlueGetCompressionType 1895  
BmpGlueGetDimensions 546, 1892  
BmpGlueGetNextBitmap 547, 1892  
BmpGlueGetTransparentValue 1895  
BmpGlueSetTransparentValue 1896  
BmpSetDensity 551  
BmpSetTransparentValue 552  
BmpSize 553  
boldFont 711, 736, 1903  
boot, and heap compacting 784  
bound of next line for global find 250  
BtLibAccessibleModeEnum 1940  
btLibActiveMode 1953  
BtLibAddrAToBtd 1930  
BtLibAddrBtdToA 1931  
btLibCachedOnly 1972  
btLibCachedThenRemote 1972  
BtLibCancelInquiry 1962  
BtLibClassOfDeviceType 1941  
BtLibClose 1960  
btLibConnectableOnly 1941  
BtLibConnectionRoleEnum 1946  
BtLibDeviceAddressType 1946  
btLibDiscoverableAndConnectable 1941  
BtLibDiscoverMultipleDevices 1963  
BtLibDiscoverSingleDevice 1966  
BtLibFriendlyNameType 1947  
BtLibGeneralPreferenceEnum 1969  
BtLibGetGeneralPreference 1968  
BtLibGetNameEnum 1971  
BtLibGetRemoteDeviceName 1970  
BtLibGetSelectedDevices 1972  
BtLibHandleEvent 2297  
BtLibHandleTransportEvent 2297  
btLibHoldMode 1953  
BtLibL2CapHToNL 1932  
BtLibL2CapHToNS 1932  
BtLibL2CapNToHL 1933  
BtLibL2CapNToHS 1933  
BtLibL2CapPsmType 1992  
BtLibLanguageBaseTripletType 1992  
BtLibLinkConnect 1973

---

BtLibLinkDisconnect 1974  
BtLibLinkGetState 1976  
BtLibLinkModeEnum 1953  
btLibLinkPref\_Authenticated 1979  
btLibLinkPref\_Encrypted 1979  
btLibLinkPref\_LinkRole 1979  
BtLibLinkPrefsEnum 1978  
BtLibLinkSetState 1977  
BtLibManagementCallback 1990  
btLibManagementEventAccessibilityChange 1949  
btLibManagementEventAclConnectInbound 1949  
btLibManagementEventAclConnectOutbound 19  
49  
btLibManagementEventAclDisconnect 1950  
btLibManagementEventAuthentication  
Complete 1950  
btLibManagementEventEncryptionChange 1950  
btLibManagementEventInquiryCanceled 1951  
btLibManagementEventInquiryComplete 1951  
btLibManagementEventInquiryResult 1951  
btLibManagementEventLocalNameChange 1951  
btLibManagementEventModeChange 1952  
btLibManagementEventNameResult 1953  
btLibManagementEventPasskeyRequested 1954  
btLibManagementEventPiconetCreated 1954  
btLibManagementEventPiconetDestroyed 1954  
btLibManagementEventRadioState 1955  
btLibManagementEventRoleChange 1956  
BtLibManagementEventType 1947  
btLibMasterRole 1946  
BtLibMutex 2297  
btLibNotAccessible 1941  
BtLibOpen 1960  
BtLibOpenBackground 2297  
btLibParkMode 1953  
BtLibPiconetCreate 1979  
BtLibPiconetDestroy 1981  
BtLibPiconetLockInbound 1982  
BtLibPiconetUnlockInbound 1983  
btLibPref\_CurrentAccessible 1969  
btLibPref\_LocalClassOfDevice 1969  
btLibPref\_Name 1969  
btLibPref\_UnconnectedAccessible 1970  
BtLibProfileDescriptorListEntryType 2003  
BtLibProtocolDescriptorListEntryType 2004  
BtLibRegisterManagementNotification 1984  
btLibRemoteOnly 1972  
BtLibRfCommHToNL 1934  
BtLibRfCommHToNS 1934  
BtLibRfCommNToHL 1935  
BtLibRfCommNToHS 1935  
BtLibRfCommServerIdType 2004  
BtLibSdpAttributeDataType 2005  
BtLibSdpAttributeIdType 2006  
BtLibSdpCompareUuids 2042  
BtLibSdpGetPSMByUuid 2043  
BtLibSdpGetRawDataElementSize 2044  
BtLibSdpGetRawDataElementType 2046  
BtLibSdpGetServerChannelByUuid 2047  
BtLibSdpHToNL 1936  
BtLibSdpHToNS 1936  
BtLibSdpNToHL 1937  
BtLibSdpNToHS 1937  
BtLibSdpParseRawDataElement 2048  
BtLibSdpRecordHandle 2007  
BtLibSdpRemoteServiceRecordHandle 2007  
BtLibSdpServiceRecordCreate 2050  
BtLibSdpServiceRecordDestroy 2051  
BtLibSdpServiceRecordGetAttribute 2052  
BtLibSdpServiceRecordGetNumList Entries 2054  
BtLibSdpServiceRecordGetNumLists 2056  
BtLibSdpServiceRecordGetRawAttribute 2058  
BtLibSdpServiceRecordGetSizeOfRaw  
Attribute 2060  
BtLibSdpServiceRecordGetStringOrURL  
Length 2062  
BtLibSdpServiceRecordMapRemote 2064  
BtLibSdpServiceRecordSetAttribute 2065  
BtLibSdpServiceRecordSetAttributesFor  
Socket 2067  
BtLibSdpServiceRecordSetRawAttribute 2068  
BtLibSdpServiceRecordsGetByService Class 2070  
BtLibSdpServiceRecordStartAdvertising 2072  
BtLibSdpServiceRecordStopAdvertising 2073  
BtLibSdpUuidInitialize 2074  
BtLibSdpUuidSizeEnum 2007  
BtLibSdpUuidType 2008  
BtLibSdpVerifyRawDataElement 2075

---

BtLibSecurityFindTrustedDeviceRecord 1926  
BtLibSecurityGetTrustedDeviceRecordInfo 1927  
BtLibSecurityNumTrustedDeviceRecords 1928  
BtLibSecurityRemoveTrustedDevice Record 1929  
BtLibServiceClose 2297  
BtLibServiceIndicateSessionStart 2297  
BtLibServiceOpen 2297  
BtLibServicePlaySound 2297  
BtLibSetGeneralPreference 1985  
btLibSlaveRole 1946  
BtLibSleep 2297  
btLibSniffMode 1953  
BtLibSocketAdvanceCredit 2024  
BtLibSocketCallback 2076  
BtLibSocketClose 2025  
BtLibSocketConnect 2026  
BtLibSocketConnectInfoType 2027  
BtLibSocketCreate 2029  
btLibSocketEventConnectedInbound 2012  
btLibSocketEventConnectedOutbound 2012  
btLibSocketEventConnectRequest 2013  
btLibSocketEventData 2013  
btLibSocketEventDisconnected 2014  
btLibSocketEventSdpGetAttribute 2014  
btLibSocketEventSdpGetNumListEntries 2017  
btLibSocketEventSdpGetNumLists 2017  
btLibSocketEventSdpGetPsmByUuid 2021  
btLibSocketEventSdpGetRawAttribute 2018  
btLibSocketEventSdpGetRawAttributeSize 2019  
btLibSocketEventSdpGetServerChannelBy  
    Uuid 2020  
btLibSocketEventSdpGetStringLen 2016  
btLibSocketEventSdpServiceRecordHandle 2014  
btLibSocketEventSendComplete 2022  
BtLibSocketEventType 2010  
BtLibSocketGetInfo 2031  
btLibSocketInfo\_L2CapChannel 2033  
btLibSocketInfo\_L2CapPsm 2033  
btLibSocketInfo\_MaxRxSize 2033  
btLibSocketInfo\_MaxTxSize 2033  
btLibSocketInfo\_Protocol 2033  
btLibSocketInfo\_RemoteDeviceAddress 2033  
btLibSocketInfo\_RfCommOutstandingCredits 203

4

btLibSocketInfo\_RfCommServerId 2034  
btLibSocketInfo\_SdpServiceRecordHandle 2034  
btLibSocketInfo\_SendPending 2034  
BtLibSocketInfoEnum 2032  
BtLibSocketListen 2034  
BtLibSocketListenInfoType 2036  
BtLibSocketRef 2010  
BtLibSocketRespondToConnection 2038  
BtLibSocketSend 2040  
BtLibStartInquiry 1987  
BtLibStringType 2011  
BtLibUnload 2297  
BtLibUnregisterManagementNotification 1988  
BtLibUrlType 2011  
BtLibWake 2297  
BufferedReuse SSL Attribute 2189  
busy bit 619  
ButtonFrameType 157  
buttons (silk-screened buttons) 56  
byteAttrFirst 998  
byteAttrLast 998  
byteAttrMiddle 998  
byteAttrSingle 998

## C

calibrating the pen 825  
canceling alarms 507  
capsLockMask 57  
card number 777  
categories, setting label 313  
category  
    DmSeekRecordInCategory 631  
    moving records 603  
Category Constants 133  
Category Data Structures 133  
Category Functions 133  
CategoryCreateList 136, 142, 2306  
CategoryCreateListV10 138  
categoryDefaultEditCategoryString 135, 137, 139,  
    147, 2329  
categoryDefaultEditText 137, 148  
CategoryEdit 139, 2306  
CategoryEditV10 141

---

CategoryEditV20 140  
CategoryFind 141  
CategoryFreeList 142, 2306  
CategoryFreeListV10 143  
CategoryGetName 144  
CategoryGetNext 145  
categoryHideEditCategory 135, 137, 148, 2329  
CategoryInitialize 134, 146  
CategorySelect 147, 2306  
CategorySelectV10 149  
CategorySetName 150  
CategorySetTriggerLabel 151  
CategoryTruncateName 152  
cCountryName constants 767  
character attribute functions 555–559  
character encodings 1003, 1021, 1028, 1035  
characters

*See Also* multi-byte characters

attributes 1000, 1004, 1005, 1006, 1008, 1009, 1010, 1011, 1013  
converting 1036  
drawable 1011  
drawing in window 1174  
erasing 1181  
graphic 1006  
inverting 1198  
printable 1009  
size 1012  
sorting text 559  
valid 1011

CharAttr.h 555  
charAttrAINum 1000  
charAttrAlpha 1000  
charAttrCntrl 1000  
charAttrDelim 1001  
charAttrGraph 1001  
charAttrPrint 1000  
charAttrSpace 1000  
charEncoding... constants 766  
CharEncodingType 765, 997  
charEncodingUnknown 1029  
checkboxFont 717  
checkboxTableItem 424  
ChrHorizEllipsis 555, 2313

ChrlsHardKey 556  
ChrNumericSpace 556, 2313  
chunks

- card number 777
- disposing of chunk 778
- heap ID 779, 793
- locking 779
- size 782
- unlocking 783, 797

CipherSuite SSL Attribute 2190  
CipherSuiteInfo SSL Attribute 2190  
CipherSuites SSL Attribute 2191  
clipboard 205, 206, 226  
Clipboard.h 153  
ClipboardAddItem 154  
ClipboardAppendItem 155  
ClipboardFormatType 153  
ClipboardGetItem 156  
Clipper application 2318  
clipping rectangle 1233  
closing net library 1424, 1438  
CncAddProfile 1273  
CncDefineParamID 1275  
CncDefineParamId 1272  
CncDeleteProfile 1277  
CncGetParamType 1272, 1278  
CncGetProfileInfo 1278  
CncGetProfileList 1280  
CncGetSystemFlagBitnum 1283  
CncGetTrueParamID 1283

CnclsFixedLengthParamType 1284  
CnclsSystemFlags 1284  
CnclsSystemRange 1285  
CnclsThirdPartiesRange 1285  
CnclsVariableLengthParamType 1286  
cncNotifyProfileEvent 75  
CncProfileCloseDB 1286  
CncProfileCount 1287  
CncProfileCreate 1275, 1288  
CncProfileDelete 1288  
CncProfileGetIDFromIndex 1263, 1290  
CncProfileGetIDFromName 1263, 1291  
CncProfileGetIndex 1291  
CncProfileID 1263

---

CncProfileNotifyDetailsType 76  
CncProfileOpenDB 1292  
CncProfileSetCurrent 1293  
CncProfileSettingGet 1270, 1273, 1294  
CncProfileSettingSet 1270, 1295  
ColorTableEntries 554  
ColorTableType 531  
commandChr 372, 381, 383, 384, 388  
commandKeyMask 57  
compacting heaps 784  
comparing memory blocks 776  
Compat SSL Attribute 2192  
compatibility 2303–??  
connect 1508  
connection manager 2322  
ConnectionMgr.h 1263  
constantRscType 501  
ContrastAdjust 490, 2327  
control objects  
    and pen tracking 62  
    drawing 167  
    erasing 168  
    selection in a group 285  
    structure 157  
Control.h 157  
ControlAttrType 158  
controlKeyMask 57  
ControlPtr 159  
ControlStyleType 160, 1897  
ControlType 44, 45, 161  
coordinates, display-relative vs. window-relative 1172  
CoreTraps.h 979, 990  
CountryType 763, 811  
CPM and AP Constants 2082  
CPMLibGenerateKey 2116  
Crc.h 1247  
Crc16CalcBlock 1247  
creating active window 1170  
creating modal window 1170  
creator ID 30  
CtlDrawControl 159, 167  
CtlEnabled 158, 168  
ctlEnterEvent 44, 45, 171, 298  
CtlEraseControl 159, 168  
ctlExitEvent 44, 45, 171  
CtlGetLabel 162, 169  
CtlGetSliderValues 169  
CtlGetValue 159, 166, 170  
CtlGlueGetFont 1897  
CtlGlueGetGraphics 1898  
CtlGlueNewSliderControl 1899  
CtlGlueSetFont 1900  
CtlGlueSetLeftAnchor 1901  
CtlHandleEvent 44, 45, 171  
CtlHideControl 158, 172  
CtlHitControl 172  
CtlNewControl 173  
CtlNewGraphicControl 173, 175  
CtlNewSliderControl 173  
ctlRepeatEvent 44, 171, 298, 2328  
ctlSelectEvent 44, 45, 298, 2328  
CtlSetEnabled 158, 178  
CtlSetGraphics 163, 164, 179  
CtlsetLabel 162, 180  
CtlSetSliderValues 181  
CtlSetUsable 158, 182  
CtlSetValue 159, 166, 182  
CtlShowControl 158, 183  
CtlValidatePointer 184  
current time 35  
custom fill patterns, getting 1192  
CustomPatternType 1147  
customTableItem 424, 425, 476

## D

data manager error codes 566–569, 594  
data storage heap 792  
    handles 778  
database ID 587  
databases  
    closing 574  
    creating 575  
    cutting and pasting 573  
    deleting. *See Also* DmDatabaseProtect  
    overlays 613  
    SysCreateDataBaseList 970  
DataMgr.h 561

---

date 188  
date system resource 185  
DateAdjust 1055  
DateDaysToDate 1056  
DateGlueTemplateToAscii 1060, 1892  
DateGlueToDOWDMFormat 1064, 1892  
DatePtr 1049  
DateSecondsToDate 1056  
dateStringLength 1061  
dateTableItem 424  
DateTemplateToAscii 1057  
DateTime.h 1045  
DateTimePtr 1048  
DateTimeType 1048  
DateToAscii 1061  
DateToDays 1062  
DateToDOWDMFormat 1063  
DateType 1049  
day selector object 46  
Day.h 185  
DayDrawDays 186  
DayDrawDaySelector 187  
DayHandleEvent 187  
DayOfMonth 1051, 1064  
DayOfWeek 1065  
DayOfWeekType 1053  
daySelectEvent 46  
DaysInMonth 1066  
debugging and MemHeapScramble 787  
debugging mode 777, 798  
defaultAlarmSoundLevel 840  
defaultAlarmSoundVolume 840  
defaultAutoLockTime 840  
defaultAutoLockTimeFlag 840  
defaultAutoLockType 840  
defaultAutoOffDuration 830, 840  
defaultAutoOffDurationSecs 835, 840  
defaultBoldFont 1903  
defaultGameSoundLevel 840  
defaultGameSoundVolume 840  
defaultLargeFont 1902  
defaultSmallFont 1902  
defaultSysSoundLevel 840  
defaultSysSoundVolume 840  
defaultSystemFont 1902  
delete bit 582, 585  
delete callback function 1352  
DeleteProc 1315, 1317, 1352  
deleting databases *See Also* DmDatabaseProtect  
deleting records 584  
DensityType 531  
DeviceInfoType 1572  
DeviceInfoType structure 1551  
dialogs  
    Edit Categories 139  
digitizer  
    and PenResetCalibration function 826  
    and penUpEvent 63  
    EvtProcessSoftKeyStroke 956  
DirectionType 229  
DlkCallAppReplyParamType 1249  
DlkControl 1248  
DlkGetSyncInfo 1251  
DLServer.h 1247  
dmAllCategories 142, 563, 611  
dmAllHdrAttrs 565  
dmAllRecAttrs 563  
DmArchiveRecord 571  
DmAttachRecord 572  
DmAttachResource 573  
dmCategoryLength 133, 144, 563  
DmCloseDatabase 574  
DmComparF 590, 602, 621, 640  
DmCreateDatabase 575  
DmCreateDatabaseFromImage 577  
DmDatabaseInfo 577  
DmDatabaseProtect 574, 580  
DmDatabaseSize 581  
dmDBNameLength 570, 575, 633  
DmDeleteCategory 582  
DmDeleteDatabase 574, 583  
DmDeleteRecord 584  
DmDetachRecord 585  
DmDetachResource 586  
dmErrDatabaseNotProtected 567  
dmErrRecordArchived 568  
DmFindDatabase 576, 584, 587

---

DmFindRecordByID 587  
DmFindResource 588  
DmFindResourceType 589  
DmFindSortPosition 590, 2306  
DmFindSortPositionV10 591  
DmGet1Resource 601, 602, 609  
DmGetAppInfoID 592  
DmGetDatabase 584, 593  
DmGetDatabaseLockState 593  
DmGetLastErr 594, 595  
DmGetNextDatabaseByTypeCreator 596  
DmGetRecord 599  
DmGetResource 600, 601  
DmGetResourceIndex 601  
dmHdrAttrAppInfoDirty 565  
dmHdrAttrBackup 565  
dmHdrAttrBundle 565  
dmHdrAttrCopyPrevention 565  
dmHdrAttrHidden 565  
dmHdrAttrLaunchableData 566  
dmHdrAttrOKToInstallNewer 566  
dmHdrAttrOpen 566  
dmHdrAttrReadOnly 566  
dmHdrAttrRecyclable 574  
dmHdrAttrResDB 566  
dmHdrAttrResetAfterInstall 566  
dmHdrAttrStream 566  
DmInsertionSort 602  
dmMaxRecordIndex 563, 572, 607  
dmModeExclusive 570  
dmModeLeaveOpen 570  
dmModeReadOnly 570  
dmModeReadWrite 570  
dmModeShowSecret 570  
dmModeWrite 570  
DmMoveCategory 603  
DmMoveRecord 605  
DmNewHandle 606  
DmNewRecord 607  
DmNewResource 608  
DmNextOpenDatabase 609  
DmNextOpenResDatabase 609  
DmNumDatabases 610  
DmNumRecords 611  
DmNumRecordsInCategory 611  
DmNumResources 612  
DmOpenDatabase 570, 613  
DmOpenDatabaseByTypeCreator 615  
DmOpenDatabaseInfo 616  
DmOpenDBNoOverlay 570, 617  
DmOpenRef 561  
DmPositionInCategory 618  
DmQueryNextInCategory 619  
DmQueryRecord 620  
DmQuickSort 621  
dmRecAttrBusy 563  
dmRecAttrCategoryMask 563, 604  
dmRecAttrDelete 563  
dmRecAttrDirty 563  
dmRecAttrSecret 563  
dmRecNumCategories 133, 563  
DmRecordInfo 622  
DmReleaseRecord 599, 607, 623  
DmReleaseResource 608, 623  
DmRemoveRecord 624  
DmRemoveResource 625  
DmRemoveSecretRecords 626  
DmResID 561  
DmResizeRecord 626  
DmResizeResource 627  
DmResourceInfo 628  
DmResType 562  
DmSearchRecord 629  
DmSearchResource 601, 629  
DmSearchStatePtr 596  
DmSearchStateType 596  
dmSeekBackward 631  
dmseekForward 631  
DmSeekRecordInCategory 631  
DmSet 632  
DmSetDatabaseInfo 633  
DmSetRecordInfo 635  
DmSetResourceInfo 636  
DmStrCopy 637  
dmSysOnlyHdrAttrs 566  
dmSysOnlyRecAttrs 563

---

dmUnfiledCategory 563, 1323  
DmWrite 638  
DmWriteCheck 639  
DontSendShutdown SSL Attribute 2193  
DontWaitForShutdown SSL Attribute 2193  
doubleTapKeyMask 57  
dowDateStringLength 1063  
dowLongDateStrLength 1063  
doze mode  
    SysTaskDelay 992  
Dragonball EZ 2313  
draw window 1235  
drawable characters 1011  
drawDetail structure 108  
drawing rectangular frame 1176, 1179, 1208  
drawItemsCallback 343, 355  
drawList structure 109  
DrawStateType 1147  
DrvEntryPoint  
    for virtual driver 1538  
DrvrInfoType structure 1523  
DrvrRcvQType structure 1525  
DrvrStatusEnum 1526  
dynamic heap  
    adding chunk 780  
    allocating chunk 794  
    moving memory 790  
    reinitializing 989  
    test 784  
dynamic heap handles 778  
dynamic scrolling 66

## E

Edit Categories dialog 139  
editingStrID 137, 147  
enabling windows 1171  
erasing characters 1181  
erasing lines in window 1182  
erasing rectangle 1183  
ErrAlert 644  
ErrCatch 645, 650  
ErrDisplay 646  
ErrDisplayFileLineMsg 647  
ErrEndCatch 647, 650

ErrExceptionList 648  
ErrExceptionType 644  
ErrFatalDisplayIf 648  
errNone 1322, 1585  
ErrNonFatalDisplayIf 649  
error code from data manager call 594  
error manager 643–650  
Error SSL Attribute 2194  
ERROR\_CHECK\_FULL 643  
ERROR\_CHECK\_LEVEL 643, 646, 649  
ERROR\_CHECK\_NONE 643  
ERROR\_CHECK\_PARTIAL 643  
ErrorBase.h 643  
ErrorMgr.h 643  
ErrThrow 650  
ErrTry 650  
event queue, adding event 942  
Event.H 941  
Event.h 39, 2328  
EventPtr 43  
events 39, 69  
eventsEnum 40  
EventType 39–69, 2327  
EvtAddEventToQueue 942  
EvtAddUniqueEventToQueue 942  
EvtCopyEvent 944  
EvtDequeuePenPoint 945  
EvtDequeuePenStrokeInfo 945  
EvtEnableGraffiti 946  
EvtEnqueueKey 946  
EvtEventAvail 947  
EvtFlushKeyQueue 948  
EvtFlushNextPenStroke 948  
EvtFlushPenQueue 949  
EvtGetEvent 62, 804, 805  
EvtGetPen 950  
EvtGetPenBtnList 951  
EvtGetPenNative 952  
EvtKeyQueueEmpty 955  
EvtKeyQueueSize 955  
EvtPenQueueSize 956  
EvtProcessSoftKeyStroke 956  
EvtResetAutoOffTimer 957

---

EvtSetAutoOffCmd 958  
EvtSysEventAvail 959  
evtWaitForever 62, 949, 1643, 1644  
EvtWakeup 960, 1591  
EvtWakeupWithoutNilEvent 960  
exchange manager 1297, 2310  
ExgAccept 21, 22, 1309, 1317, 1320, 1342  
exgAskCancel 18  
exgAskOk 18  
ExgAskParamType 19  
ExgAskResultType 1297  
exgBeamPrefix 1308  
exgBeamScheme 1308, 1309  
ExgConnect 1310, 1320  
ExgControl 1313  
ExgDBDeleteProcPtr 659, 1122, 1315  
ExgDBRead 1315, 1353  
ExgDBReadProcPtr 1353  
ExgDBWrite 1318  
ExgDBWriteProcPtr 660, 1129, 1318  
ExgDialogInfoType 1322, 1323  
ExgDisconnect 19, 21, 1312, 1319, 1326, 1340, 1342, 1350  
ExgDoDialog 18, 1298, 1322, 1335, 1337  
exgErrNoKnownTarget 1327, 1332, 1333, 1335, 1337, 1338, 1351  
exgErrNotSupported 1302, 1335  
ExgGet 1312, 1320, 1325, 1326, 1342  
exgGet 1337  
ExgGetDefaultApplication 1327  
ExgGetRegisteredApplications 1328  
ExgGetRegisteredTypes 1330  
ExgGetTargetApplication 1331, 1336, 1337  
ExgGoToType 1298  
ExgLibAccept 1357  
exgLibAPIVersion 1314  
ExgLibClose 1358  
ExgLibConnect 1359  
ExgLibControl 1360  
exgLibCtlGetPreview 1314  
exgLibCtlGetTitle 1314  
exgLibCtlGetVersion 1314  
exgLibCtlSpecificOp 1315  
ExgLibDisconnect 1362  
ExgLibGet 1363  
ExgLibHandleEvent 1364  
ExgLibOpen 1365  
ExgLibPut 1366  
ExgLibReceive 1368  
ExgLibRequest 1369  
ExgLibSend 1370  
ExgLibSleep 1371  
exgLibSmsIncompleteDeleteOp 2233  
exgLibSmsIncompleteGetCountOp 2233  
exgLibSmsPrefDisplayOp 2233  
exgLibSmsPrefGetDefaultOp 2233  
exgLibSmsPrefGetOp 2233  
exgLibSmsPrefSetOp 2233  
ExgLibWake 1372  
ExgLocalLib.h 1297  
exgLocalOpAccept 1300  
exgLocalOpGet 1300  
exgLocalOpGetSender 1300  
exgLocalOpNone 1300  
exgLocalOpPut 1300  
ExgLocalOpType 1299  
exgLocalPrefix 1305, 1309  
exgLocalScheme 1308, 1309  
ExgLocalSocketInfoType 1299  
exgMaxDescriptionLength 1329, 1343  
exgMaxTitleLen 1329  
exgMaxTypeLength 1329, 1330, 1343  
exgMemError 1331, 1344  
ExgMgr.h 653, 1297  
exgNoAsk 1337  
ExgNotifyGoto 1320, 1321, 1334, 1339  
ExgNotifyPreview 1301, 1323, 1335  
ExgNotifyReceive 1333, 1334, 1336  
exgPreviewDialog 20, 1301  
exgPreviewDraw 20, 1301  
exgPreviewFirstUser 21, 1302  
ExgPreviewInfoType 1300, 1301, 1335  
exgPreviewLastUser 21, 1302  
exgPreviewLongString 20, 1301  
exgPreviewQuery 20, 1302  
exgPreviewShortString 1302  
ExgPut 19, 1312, 1319, 1320, 1339, 1350

---

ExgReceive 21, 22, 1312, 1317, 1326, 1341, 1353  
exgRegCreatorID 1307, 1345  
exgRegExtensionID 1307, 1327  
ExgRegisterData 28  
ExgRegisterDatatype 1342  
ExgRegisterDataV35 1347  
exgRegSchemeID 1307, 1328, 1346  
exgRegTypeID 1307, 1345  
ExgRequest 27, 1348  
ExgSend 19, 1312, 1319, 1340, 1349, 1355  
exgSendBeamPrefix 1309  
exgSendPrefix 1308  
exgSendScheme 1308, 1309, 1346  
ExgSetDefaultApplication 1345, 1346, 1350  
ExgSocketType 19, 656, 1298, 1303, 1315, 1318  
exgTitleBufferSize 1314  
exgUnwrap 1337, 1338, 1344  
Expansion Manager 79, 80, 653  
expCapabilityHasStorage 654  
expCapabilityReadOnly 654  
expCapabilitySerial 654  
ExpCardGetSerialPort 656  
ExpCardInfo 657  
ExpCardInfoType 653  
ExpCardPresent 658  
expErrCardBadSector 655  
expErrCardNoSectorReadWrite 655  
expErrCardNotPresent 654  
expErrCardProtectedSector 655  
expErrCardReadOnly 655  
expErrEnumerationEmpty 655  
expErrIncompatibleAPIVer 655  
expErrInvalidSlotRefNum 655  
expErrNotEnoughPower 654  
expErrNotOpen 655  
expErrSlotDeallocated 655  
expErrStillOpen 655  
expErrUnimplemented 655  
expErrUnsupportedOperation 654  
expHandledSound 80  
expHandledVolume 80  
expMediaType\_Any 656  
expMediaType\_CompactFlash 656

expMediaType\_MacSim 656  
expMediaType\_MemoryStick 656  
expMediaType\_MultiMediaCard 656  
expMediaType\_PoserHost 656  
expMediaType\_RAMDisk 656  
expMediaType\_SecureDigital 656  
expMediaType\_SmartMedia 656  
ExpSlotDriverInstall 659  
ExpSlotDriverRemove 660  
ExpSlotEnumerate 661  
ExpSlotLibFind 662  
extended font resource (nfnt) 721  
extended gadget 259, 318, 327  
EZ Dragonball 2313

## F

fatal alert 418  
FatalAlert.h 417  
fcntl 1501  
FeatureMgr.h 665  
features *See* functions starting with Ftr  
fgetc 899  
fgets 900  
field objects  
    and text height 218  
    modifying 208  
    structure 198  
Field.h 195  
FieldAttrType 195  
FieldPtr 198  
FieldType 198  
file mode constants 673, 674  
file streaming 2309  
FileClearerr 675  
FileClose 675  
FileControl 676  
FileDelete 680  
FileDmRead 681  
FileEOF 682  
FileError 683  
FileFlush 683  
FileGetLastError 684  
FileInfoType 1075

---

FileOpen 685  
FileOpEnum 677  
FileOriginEnum 689  
FileRead 687  
FileRef 1076  
FileRewind 688  
FileSeek 688  
FileStream.h 673  
FileTell 689  
FileTruncate 690  
FileWrite 691  
fill patterns  
    getting 1192  
    setting 1238  
Find (global find) 22, 26, 249–253  
    saving data 33  
Find (lookup) 31  
Find icon 56  
Find.h 249  
FindDrawHeader 25, 249  
FindGetLineBounds 250  
FindParamsType 22, 23  
FindSaveMatch 24, 27, 250  
FindStrInStr 24, 252  
flags, launch flags 36  
FldCalcFieldHeight 204  
fldChangedEvent 47, 230, 2306  
FldCompactText 205  
FldCopy 205  
FldCut 206  
FldDelete 207  
FldDirty 208  
FldDrawField 208  
fldEnterEvent 47, 221, 298  
FldEraseField 209  
FldFreeMemory 210  
FldGetAttributes 211  
FldGetBounds 211  
FldGetFont 212  
FldGetInsPtPosition 212  
FldGetMaxChars 213  
FldGetNumberOfBlankLines 213  
FldGetScrollPosition 214  
FldGetScrollValues 214  
FldGetSelection 215  
FldGetTextAllocatedSize 216  
FldGetTextHandle 217  
FldGetTextHeight 218  
FldGetTextLength 219  
FldGetTextPtr 219  
FldGetVisibleLines 220  
FldGrabFocus 220, 446  
FldHandleEvent 47, 48, 221  
fldHeightChangedEvent 48, 223, 231, 298  
FldInsert 222  
FldMakeFullyVisible 223  
FldNewField 224  
FldPaste 226  
FldRecalculateField 227  
FldReleaseFocus 228  
FldScrollable 228  
FldScrollView 229  
FldSendChangeNotification 230  
FldSendHeightChangeNotification 231  
FldSetAttributes 231  
FldSetBounds 232  
FldSetDirty 234  
FldSetFont 235  
FldSetInsertionPoint 235  
FldSetInsPtPosition 236  
FldSetMaxChars 237  
FldSetMaxVisibleLines 237  
FldSetScrollPosition 238  
FldSetSelection 239  
FldSetText 240  
FldSetTextAllocatedSize 242  
FldSetTextHandle 243  
FldSetTextPtr 245  
FldSetUsable 246  
FldUndo 246  
FldWordWrap 247  
FloatMgr.h 695  
FloatMgr.h (Palm OS 1.0) 2355  
FloatMgrOld.h 2355  
flushing pen queue 949  
fntAppCustomBase 712  
FntAverageCharWidth 713, 716, 723

---

FntBaseLine 714, 716, 723  
FntCharHeight 713, 716, 724  
FntCharsInWidth 724  
FntCharsWidth 725  
FntCharWidth 710, 714, 716, 726  
FntDefineFont 727  
FntDescenderHeight 714, 716, 728  
fntExtendedFormatMask 715, 717  
FntGetFont 711, 728  
FntGetFontPtr 729  
FntGetScrollValues 729  
FntGlueGetDefaultFontID 736, 1902  
FntGlueWCharWidth 726, 733, 1892  
FntGlueWidthToOffset 734, 1892  
FntLineHeight 714, 716, 731  
FntLineWidth 731  
fntMissingChar 717  
FntSetFont 711, 732  
fntTabChrWidth 717  
FntWCharWidth 710, 714, 716, 726, 732  
FntWidthToOffset 725, 733  
FntWordWrap 734  
FntWordWrapReverseNLines 735  
focus  
    and modal window 1201  
    FrmGetFocus 287  
    FrmSetFocus 316  
font resource (NFNT) 718  
font resource, extended (nfnt) 721  
Font.h 709  
FontCharInfoPtr 709  
FontCharInfoType 709, 714, 716  
FontDefaultType 1902  
FontDensityType 710, 717  
FontID 711, 727  
FontPtr 712  
fonts  
    and FldGetFont 212  
FontSelect 736  
FontSelect.h 709  
FontType 712, 719  
FontTypeV2 710, 714, 721  
form objects  
    FormType structure 271  
                functions 274–325  
    form, active 284, 285  
    Form.h 255  
    FormActiveStateType 310, 312  
    FormAttrType 255  
    FormBitmapType 257  
    FormCheckResponseFuncType 274, 279, 325  
    FormEventHandlerType 272, 327  
    FormFrameType 257  
    FormGadgetAttrType 258  
    formGadgetDeleteCmd 280, 283, 327  
    formGadgetDrawCmd 322, 328  
    formGadgetEraseCmd 301, 328  
    formGadgetHandleEventCmd 328  
    FormGadgetHandlerType 49, 50, 280, 283, 301,  
        322, 327  
    FormGadgetType 49, 51, 261, 327  
    FormGadgetTypeInCallback 262  
    FormLabelType 263  
    FormLineType 264  
    FormObjAttrType 265  
    FormObjectKind 265  
    FormObjectType 267  
    FormObjListType 268  
    FormPopupType 269  
    FormPtr 270  
    FormRectangleType 270  
    FormTitleType 271  
    FormType 271, 1906  
    FplAdd 2355  
    FplAToF 2356  
    FplBase10Info 2356  
    FplDiv 2357  
    FplFloatToLong 2357  
    FplFloatToULong 2358  
    FplFree 2358  
    FplFToA 2359  
    FplInit 2359  
    FplLongToFloat 2360  
    FplMul 2360  
    FplSub 2360  
    fprintf 900  
    fputc 901

---

fputs 902  
frame type constants 1152  
FrameBitsType 1151  
frames  
  drawing in window 1176, 1179, 1208  
FrameType 1152  
FrmAlert 275  
FrmCloseAllForms 49, 276  
frmCloseEvent 49, 276, 296, 298  
FrmCopyLabel 276  
FrmCopyTitle 277  
FrmCustomAlert 278  
FrmCustomResponseAlert 279  
FrmDeleteForm 280, 327  
FrmDispatchEvent 281, 297  
FrmDoDialog 281  
FrmDrawForm 282, 299, 328  
FrmEraseForm 54, 283  
frmGadgetEnterEvent 49, 298, 329  
frmGadgetMiscEvent 50, 298, 329  
FrmGetActiveField 284  
FrmGetActiveForm 284  
FrmGetActiveFormID 285  
FrmGetControlGroupSelection 285  
FrmGetControlValue 286  
FrmGetFirstForm 287  
FrmGetFocus 287  
FrmGetFormBounds 288  
FrmGetFormId 288  
FrmGetFormPtr 289  
FrmGetGadgetData 261, 289  
FrmGetLabel 290  
FrmGetNumberOfObjects 290  
FrmGetObjectBounds 162, 163, 165, 291  
FrmGetObjectID 291  
FrmGetObjectIndex 292  
FrmGetObjectPosition 294  
FrmGetObjectPtr 294  
FrmGetObjectType 295  
FrmGetTitle 295  
FrmGetWindowHandle 296  
FrmGlueGetActiveField 1904  
FrmGlueGetDefaultButtonID 1904  
FrmGlueGetHelpID 1905  
FrmGlueGetLabelFont 1905  
FrmGlueGetMenuBarID 1906  
FrmGlueSetDefaultButtonID 1907  
FrmGlueSetHelpID 1907  
FrmGlueSetLabelFont 1908  
frmGotoEvent 51  
FrmGotoForm 49, 52, 296  
FrmGraffitiStateType 273  
FrmHandleEvent 49, 53, 63, 297, 328  
FrmHelp 300  
FrmHideObject 169, 172, 259, 301, 328  
FrmInitForm 302  
frmInvalidObjectId 292  
frmLoadEvent 52, 296  
FrmNewBitmap 303  
FrmNewForm 304  
FrmNewGadget 305  
FrmNewGsi 306  
FrmNewLabel 307  
frmNoSelectedControl 274, 285  
frmOpenEvent 51, 52, 272, 281, 282, 296, 309  
FrmPointInTitle 308  
FrmPopupForm 52, 309  
frmRedrawUpdateCode 55, 274, 322  
FrmRemoveObject 309  
frmResponseCreate 274, 326  
frmResponseQuit 274, 326  
FrmRestoreActiveState 310, 312  
FrmReturnToForm 311  
FrmSaveActiveState 310, 312  
FrmSaveAllForms 53, 312  
frmSaveEvent 53, 312  
FrmSetActiveForm 68, 313  
FrmSetCategoryLabel 313  
FrmSetControlGroupSelection 314  
FrmSetControlValue 315  
FrmSetEventHandler 316  
FrmSetFocus 316, 446  
FrmSetGadgetData 261, 317  
FrmSetGadgetHandler 261, 318  
FrmSetMenu 319, 379, 385, 386  
FrmSetObjectBounds 162, 163, 165, 319

---

FrmSetObjectPosition 320  
FrmSetTitle 320  
FrmShowObject 184, 259, 321, 328  
frmTitleEnterEvent 53, 298  
frmTitleSelectEvent 53, 298, 2328  
frmUpdateEvent 54, 274, 282, 299, 322  
FrmUpdateForm 54, 55, 322  
FrmUpdateScrollers 323  
FrmValidatePtr 323  
FrmVisible 325  
ftrErrNoSuchFeature 665, 666, 667, 669, 670, 671  
FtrGet 665  
FtrGetByIndex 666  
FtrPtrFree 667  
FtrPtrNew 667  
FtrPtrResize 669  
FtrSet 670  
FtrUnregister 670

**G**

gadget resource 259, 317  
    extended 259, 318, 327  
getchar 902  
GetCharAttr 557  
GetCharCaselessValue 558  
GetCharSortValue 559  
gethostname 1481  
gets 903  
GetSize 1546  
GetSpace 1547  
global find 22, 26, 249–253, 1022, 1914, 1918  
    FindDrawHeader 249  
    FindGetLineBounds 250  
    saving data 33  
gotIt structure 110  
goto (global find) 26  
GoToParamsType 26  
Graffiti  
    Command shortcuts 60  
    enabling and disabling 946  
Graffiti manager  
    functions 737–747  
Graffiti recognizer  
    EvtDequeuePenPoint 945

Graffiti Reference Dialog 418  
Graffiti Shift  
    functions 331–333  
Graffiti.h 737  
GraffitiReference.h 417  
GraffitiShift.h 331  
GraffitiUI.h 417  
graphic characters 1006  
GraphicControlType 162, 167  
GraphicStatePtr 1159  
GrfAddMacro 737  
GrfAddPoint 738  
GrfCleanState 738  
GrfDeleteMacro 739  
GrfFilterPoints 739  
GrfFindBranch 740  
GrfFlushPoints 740  
GrfGetAndExpandMacro 741  
GrfGetGlyphMapping 741  
GrfGetMacro 742  
GrfGetMacroName 742  
GrfGetNumPoints 743  
GrfGetPoint 743  
GrfGetState 744  
GrfInitState 745  
GrfMatch 745  
GrfMatchGlyph 746  
GrfProcessStroke 746  
GrfSetState 747  
groups of controls 285  
GsiEnable 331  
GsiEnabled 332  
GsiInitialize 332  
GsiSetLocation 332  
GsiSetShiftState 333

**H**

hard reset 35  
header line for global find 249  
heap ID 786, 793  
    of chunk 779  
heaps  
    compacting 784

---

free bytes 785  
ROM based 785  
height of text in field 218  
Helper API 749  
Helper.h 749  
HelperNotifyEnumerateListType 749  
HelperNotifyEventType 749, 751, 754  
HelperNotifyExecuteType 752, 755, 756  
HelperNotifyValidateType 754  
HelperServiceClass.h 749  
HelperServiceEMailDetailsType 754, 755  
HelperServiceSMSDetailsType 754  
HelperServiceSMSDetailType 756  
HostControl.h 980  
hostent 1414  
HostGremlinIsRunning 980  
HotSync and sysAppLaunchCmdSyncNotify 34  
HotSync operation 101  
HsState SSL Attribute 2194

**I**

icons 56  
iconType 535  
ID  
  databases 587  
  heap 786  
iMessenger application 2318  
IndexedColorType 1153  
INetCacheEntryType 1856  
INetCacheInfoType 1854  
inetCfgName... constants 1850–1851  
inetCompressionType... constants 1840  
INetCompressionTypeEnum 1839  
INetConfigNameType 1840  
inetContentType... constants 1841  
INetContentTypeEnum 1841  
inetHTTPAttr... constants 1843–1844  
INetHTTPAttrEnum 1841  
INetLibCacheGetObject 1853  
INetLibCacheList 1854  
INetLibCheckAntennaState 1856  
INetLibClose 1857  
INetLibConfigAliasGet 1858

INetLibConfigAliasSet 1859  
INetLibConfigDelete 1860  
INetLibConfigIndexFromName 1861  
INetLibConfigList 1862  
INetLibConfigMakeActive 1863  
INetLibConfigRename 1864  
INetLibConfigSaveAs 1865  
INetLibGetEvent 1866  
INetLibOpen 1867  
INetLibSettingGet 1868  
INetLibSettingSet 1869  
INetLibSockClose 1870  
INetLibSockConnect 1871  
INetLibSockHTTPAttrGet 1872  
INetLibSockHTTPAttrSet 1873  
INetLibSockHTTPReqCreate 1874  
INetLibSockHTTPReqSend 1875  
INetLibSockOpen 1877  
INetLibSockRead 1878  
INetLibSockSettingGet 1879  
INetLibSockSettingSet 1880  
INetLibSockStatus 1881  
INetLibURLCrack 1882  
INetLibURLGetInfo 1884  
INetLibURLOpen 1885  
INetLibURLsAdd 1886  
INetLibWiCmd 1888  
INetMgr.h 39, 1839, 2237  
inetOpenURLFlag... constants 1852  
inetScheme... constants 1844–1845  
INetSchemeEnum 1844  
inetSetting... constants 1846–1847  
INetSettingEnum 1845  
inetSockReadyEvent 55  
inetSockSetting... constants 1848–1849  
INetSockSettingEnum 1847  
inetSockStatusChangeEvent 56  
inetStatus... constants 1850  
INetStatusEnum 1849  
InetURLInfo type 1884  
inetURLInfoFlag... constants 1852  
InetURLType 1883  
InfoInterest SSL Attribute 2196

---

initialization 30  
insertion point functions 335–338  
insertion points  
    and FldGetInsPtPosition 212  
    and FldGrabFocus 220  
    and FldReleaseFocus 228  
    and FldSetInsertionPoint 235  
    displayed in field 209  
insertion sort 983  
InsPoint.h 335  
InsPtEnable 335  
InsPtEnabled 336  
InsPtGetHeight 336  
InsPtGetLocation 337  
InsPtSetHeight 337  
InsPtSetLocation 338  
international manager 2316  
Internet library 1839  
intlErrInvalidSelector 1256  
IntlGetRoutineAddress 1255, 1256  
IntlGlue.h 1909  
IntlGlueGetRoutineAddress 1908  
IntlMgr.h 1247, 1255, 1256, 1908  
IntlSetRoutineAddress 1256  
inverting characters in draw window 1198  
inverting line in draw window 1199  
IoFlags SSL Attribute 2197  
IoStruct SSL Attribute 2198  
IoTimeout SSL Attribute 2197  
IR Library 2310  
IR manager 1373  
IrAdvanceCredit 1383  
IrBind 1384  
IrCallbackParms 1377  
IrClose 1385  
IrConnectIrLap 1385  
IrConnectReq 1386  
IrConnectRsp 1388  
IrDataReq 1389  
IrDisconnectIrLap 1390  
IrDiscoverReq 1391  
irGetScanningMode 1378  
irGetStatistics 1378  
IrIAS\_Add 1400  
IrIAS\_GetInteger 1401  
IrIAS\_GetIntLsap 1401  
IrIAS\_GetObjectID 1402  
IrIAS\_GetOctetString 1403  
IrIAS\_GetOctetStringLen 1403  
IrIAS.GetType 1404  
IrIAS.GetUserString 1404  
IrIAS.GetUserStringCharSet 1404  
IrIAS.GetUserStringLen 1405  
IrIAS\_Next 1406  
IrIAS\_Query 1406  
IrIAS\_SetDeviceName 1408  
IrIAS\_StartResult 1409  
IrIsQueryCallback 1409  
IrIsIrLapConnected 1392  
IrIsMediaBusy 1392  
IrIsNoProgress 1393  
IrIsRemoteBusy 1393  
irlib.h 1373  
IrLocalBusy 1393  
IrMaxRxSize 1394  
IrMaxTxSize 1395  
IrOpen 1395  
irRestoreScanning 1378  
irSetBaudMask 1379  
IrSetConTypeLMP 1396  
IrSetConTypeTTP 1397  
IrSetDeviceInfo 1397  
irSetScanningMode 1379  
irSetSerialMode 1380  
irSetSupported 1380  
irSuppressScanning 1380  
IrTestReq 1398  
IrUnbind 1399  
ISO 639 767  
iterate structure 111

## J

Japanese feature set 2318

## K

kAttnCommandCustomEffect 106  
kAttnCommandDrawDetail 106

---

kAttnCommandDrawList 106  
kAttnCommandGoThere 106  
kAttnCommandGotIt 106  
kAttnCommandIterate 107  
kAttnCommandPlaySound 106  
kAttnCommandSnooze 107  
kAttnFlagsAllBits 112  
kAttnFlagsAlwaysCustomEffect 112  
kAttnFlagsAlwaysLED 112  
kAttnFlagsAlwaysSound 112  
kAttnFlagsAlwaysVibrate 112  
kAttnFlagsCapabilitiesMask 117  
kAttnFlagsCustomEffectBit 112  
kAttnFlagsEverything 113  
kAttnFlagsHasCustomEffect 118  
kAttnFlagsHasLED 118  
kAttnFlagsHasSound 118  
kAttnFlagsHasVibrate 118  
kAttnFlagsLEDBit 112  
kAttnFlagsNoCustomEffect 113  
kAttnFlagsNoLED 113  
kAttnFlagsNoSound 113  
kAttnFlagsNothing 113  
kAttnFlagsNoVibrate 113  
kAttnFlagsSoundBit 112  
kAttnFlagsUserSettingsMask 117  
kAttnFlagsUserWantsCustomEffect 118, 837  
kAttnFlagsUserWantsLED 118, 837  
kAttnFlagsUserWantsSound 118, 837  
kAttnFlagsUserWantsVibrate 118, 837  
kAttnFlagsUseUserSettings 112  
kAttnFlagsVibrateBit 112  
kAttnFtrCapabilities 117  
kAttnFtrCreator 117  
kAttnLevelInsistent 115  
kAttnLevelSubtle 115  
kCncDeviceKindLocalNetwork 1272  
kCncDeviceKindModem 1272  
kCncDeviceKindPhone 1272  
kCncDeviceKindSerial 1272  
kCncErrDBAccessFailed 1289, 1290, 1291, 1292, 1293, 1294, 1295  
kCncErrProfileParamNotFound 1289, 1290, 1291, 1292, 1296  
kCncFtrCncMgrVersion 2323  
kCncMgrVersion 2323  
kCncNotifyCreateRequest 76  
kCncNotifyDeleteRequest 76  
kCncNotifyModifyRequest 76  
kCncNotifyUpdateListRequest 77  
kCncParam\_PSDCreator 1268, 1271  
kCncParam\_PSDName 1268, 1271  
kCncParam\_PSDParameterBuffer 1268  
kCncParam\_PSDType 1268, 1271  
kCncParamBaud 1264, 1270  
kCncParamBluetoothDeviceAddr 1264, 1270  
kCncParamBluetoothDeviceName 1264, 1270  
kCncParamBuffer 1273  
kCncParamCountryIndex 1264, 1265, 1270  
kCncParamDeviceKind 1264, 1265, 1271  
kCncParamDialingMode 1264, 1265, 1271  
kCncParamFlowControl 1271  
kCncParamInitString 1271  
kCncParamIntlModemCountryStringList 1266  
kCncParamIntlModemResetStringList 1266, 1269  
kCncParamInvisible 1267, 1271  
kCncParamLocked 1267, 1271  
kCncParamName 1267, 1271  
kCncParamNoDetails 1267, 1271  
kCncParamNonEditable 1268, 1271  
kCncParamPort 1268, 1271  
kCncParamReadOnly 1269, 1271  
kCncParamReceiveTimeOut 1269, 1271  
kCncParamResetString 1269, 1271  
kCncParamSerialPortFlags 1269  
kCncParamString 1273  
kCncParamSystemFlag 1273, 1276  
kCncParamSystemFlags 1269, 1271, 1273  
kCncParamThirdPartiesRange 1275  
kCncParamTimeOut 1269, 1271  
kCncParamTTCreator 1270, 1272  
kCncParamTTType 1270, 1272  
kCncParamUInt16 1273  
kCncParamUInt32 1273  
kCncParamUInt8 1273  
kCncParamVersion 1270, 1272  
kCncParamVolume 1270, 1272

---

kCncProfileClassicResetStringSize 1271  
kCncProfileNameSize 1271  
kCncProfileUsualInitStringSize 1271  
kCncProfileUsualResetStringSize 1271  
kCncProfileVersion 1270  
kCoordinatesDouble 1163  
kCoordinatesNative 1163  
kCoordinatesOneAndAHalf 1163  
kCoordinatesQuadruple 1163  
kCoordinatesStandard 1163  
kCoordinatesTriple 1163  
kDensityDouble 710, 714  
kDensityLow 710, 714  
kDrvrVersion 1535  
kDrvrVersion3 1535  
kDrvrVersion4 1535  
key manager functions 759–761  
key queue  
    size 955  
keyBitPageDown 759  
keyBitPageUp 759  
keyBitPower 759  
keyboard display 984  
KeyCurrentState 759  
keyDownEvent 56, 99, 221, 272, 299, 362, 375, 382,  
    383, 946, 2314  
KeyMgr.h 759  
KeyRates 760  
KeySetMask 761  
kHelperNotifyActionCodeEnumerate 749, 752  
kHelperNotifyActionCodeExecute 752  
kHelperNotifyActionCodeValidate 752, 754  
kHelperServiceClassIDEMail 754, 755, 757  
kHelperServiceClassIDFax 754, 757  
kHelperServiceClassIDSMS 754, 756, 757  
kHelperServiceClassIDVoiceDial 754, 757  
kMaxCountryNameLen 768, 770  
kMaxCurrencyNameLen 768, 770  
kMaxCurrencySymbolLen 768, 770, 771

**L**

labelTableItem 425  
LanguageType 763, 811

largeBoldFont 712, 736, 1903, 2308  
largeFont 711, 736, 1903  
LastAlert SSL Attribute 2198  
LastApi SSL Attribute 2199  
LastIO SSL Attribute 2200  
launch codes  
    summary 3, 71, 749  
    SysBroadcastActionCode 968  
launch flags 36  
Launcher.h 417  
ledFont 712  
LEVENT\_DATA\_IND 1381  
LEVENT\_DISCOVERY\_CNF 1381  
LEVENT\_LAP\_CON\_CNF 1381  
LEVENT\_LAP\_CON\_IND 1381  
LEVENT\_LAP\_DISCON\_IND 1381  
LEVENT\_LM\_CON\_CNF 1381  
LEVENT\_LM\_CON\_IND 1381  
LEVENT\_LM\_DISCON\_IND 1382  
LEVENT\_PACKET\_HANDLED 1382  
LEVENT\_STATUS\_IND 1382  
LEVENT\_TEST\_CNF 1382  
LEVENT\_TEST\_IND 1382  
libEvtHookKeyMask 57  
libPalmOSGlue.a 1891  
LineInfoPtr 202  
LineInfoType 203  
lines  
    erasing 1182  
    inverting 1199  
list objects  
    and pen tracking 62  
    creating category list 136  
    drawItemsCallback 343, 355  
    fields 341  
    functions 343–355  
    structure 340  
List.h 339  
ListAttrType 339  
ListDrawDataFuncType 355  
ListPtr 343  
lists  
    setting items 974  
    ListType structure 340

---

lLanguageName constants 767  
lmAnyCountry 773  
lmAnyLanguage 194, 773, 1071  
lmChoice... constants 770  
lmErrBadLocaleIndex 769  
lmErrBadLocaleSettingChoice 769  
lmErrSettingDataOverflow 769  
lmErrUnknownLocale 773  
LmGetLocaleSetting 772, 773  
LmGlueGetLocaleSetting 771, 1892  
LmGlueGetNumLocales 772, 1892  
LmGlueLocaleToIndex 774, 1892  
LmLocaleType 764, 770, 773, 837  
local ID 788, 796  
    from chunk handle 782  
locale 837  
LocaleMgr.h 763  
Localize.h 763, 1247  
LocGetNumberSeparators 16, 765, 1257  
locking chunk 779  
locking system 34  
longDateStrLength 1061  
lookup 31  
    example 31  
LstDrawList 343  
lstEnterEvent 57, 58, 299, 347  
LstEraseList 344  
lstExitEvent 58  
LstGetNumberOfItems 344  
LstGetSelection 345  
LstGetSelectionText 345  
LstGetTopItem 346  
LstGetVisibleItems 346  
LstGlueGetFont 1910  
LstGlueGetItemsText 1910  
LstGlueGetTopItem 1892  
LstGlueSetFont 1911  
LstGlueSetIncrementalSearch 1911  
LstHandleEvent 57, 58, 347  
LstMakeItemVisible 348  
LstNewList 349  
LstPopupList 350  
LstScrollList 351  
lstSelectEvent 58

LstSetDrawFunction 351  
LstSetHeight 352  
LstSetListChoices 352  
LstSetPosition 353  
LstSetSelection 354  
LstSetTopItem 355

**M**

maxFieldLines 224  
maxFieldTextLen 237  
maxStrIToALen 925  
MdmDial 1411  
mdmErrBusy 1411  
mdmErrCmdError 1411  
mdmErrNoDCD 1411  
mdmErrNoTone 1411  
mdmErrUserCan 1411  
MdmHangUp 1412  
MeasurementSystemType 770  
MemCardInfo 775  
MemCmp 776  
MemDebugMode 777  
memErrChunkLocked 781  
memErrInvalidParam 668, 669, 780, 783, 793  
memErrNotEnoughSpace 155, 668, 669, 670, 780,  
    962, 968, 1464, 1600  
MemHandleCardNo 777  
MemHandleDataStorage 778  
MemHandleFree 778  
MemHandleHeapID 779  
MemHandleLock 779  
MemHandleNew 780  
MemHandleResize 780  
MemHandleSetOwner 781  
MemHandleSize 782  
MemHandleSsetOwner 781  
MemHandleToLocalID 782  
MemHandleUnlock 783  
MemHeapCheck 783, 784  
MemHeapCompact 784  
MemHeapDynamic 784  
memHeapFlagReadOnly 785  
MemHeapFlags 785

---

MemHeapFreeBytes 785  
MemHeapID 786  
MemHeapScramble 787  
MemHeapSize 787  
MemLocalIDKind 788  
MemLocalIDToGlobal 788  
MemLocalIDToGlobalNear 788  
MemLocalIDToLockedPtr 789  
MemLocalIDToPtr 789  
MemMove 790  
MemNumCards 790  
MemNumHeaps 786, 791  
MemNumRAMHeaps 791  
memory  
  and FldCompactText 205  
  and FldFreeMemory 210  
  and FldSetText 241, 244  
memory blocks, comparing 776  
memory card information 775  
memory manager  
  debugging mode 777, 798  
MemoryMgr.h 775  
MemPtrCardNo 792  
MemPtrDataStorage 792  
MemPtrFree 793  
MemPtrHeapID 793  
MemPtrNew 794  
MemPtrRecoverHandle 794  
MemPtrResize 795  
MemPtrSetOwner 795  
MemPtrSize 796  
MemPtrSsetOwner 795  
MemPtrToLocalID 796  
MemPtrUnlock 797  
MemSet 797  
MemSetDebugMode 798  
MemStoreInfo 799  
Menu Item Object fields 368  
menu objects  
  *See Also* menus 365  
  fields 365  
  structure 365  
menu pulldown object 369  
Menu.h 359  
MenuBarAttrType 359  
MenuBarPtr 364  
MenuBarType 365, 372  
menuButtonCause 61, 383  
menuChr 372, 381, 383, 384, 388  
menuCloseEvent 59  
MenuCmdBarAddButton 59, 95, 361, 372, 373  
MenuCmdBarButtonType 360, 362  
MenuCmdBarDisplay 376  
MenuCmdBarGetButtonData 376  
menuCmdBarMaxTextLength 377  
menuCmdBarOnLeft 372  
menuCmdBarOnRight 372  
menuCmdBarOpenEvent 59, 95, 222, 299, 373, 376, 377, 383  
menuCmdBarResultMenuItem 375  
MenuCmdBarResultType 362  
MenuCmdBarType 360, 363, 366  
menuCommandCause 61, 383  
MenuDispose 378  
MenuDrawMenu 360, 378  
MenuEraseStatus 363, 377, 380  
menuErrNoMenu 371  
menuErrNotFound 372  
menuErrOutOfMemory 373  
menuErrSameId 372  
menuErrTooManyItems 373  
menuEvent 54, 60, 299, 362, 375, 382, 383  
MenuGetActiveMenu 381  
MenuHandleEvent 59, 60, 95, 360, 375, 378, 382  
MenuHideItem 384  
MenuInit 385  
MenuItemType 367, 372  
menuOpenEvent 60, 372, 383, 384, 388  
MenuPullDownPtr 369  
MenuPullDownType 367, 369  
menus  
  FrmSetMenu 319  
  functions 371–386  
  MenuSeparatorChar 368, 370, 371  
  MenuSetActiveMenu 385  
  MenuSetActiveMenuRscID 386  
  MenuShowItem 387

---

missing character symbol 725  
modal window 350, 1170, 1201  
Mode SSL Attribute 2200  
modem 1411  
ModemMgr.h 1411  
modified field objects 208  
multi-byte characters 998, 1001, 1023, 1024, 1025, 1030, 1032  
    attributes 1013  
    comparison 999, 1014  
    converting 1036  
    encodings support 997–1040  
    searching 1022, 1914, 1918  
    size 1012  
multiple preferences 847

## N

narrowTextTableItem 427, 433, 434, 451  
net library  
    closing 1424, 1438  
    open count 1466  
    opening 1463, 1464  
netCfgNameCTPWireless 1420  
netCfgNameCTPWireline 1420  
netCfgNameDefault 1420  
netCfgNameDefWireless 1420  
netCfgNameDefWireline 1420  
netConfigIndexCurSettings 1466  
NetConfigNamePtr 1413  
NetConfigNameType 1413  
netErrAlreadyOpen 1463, 1465  
netErrAlreadyOpenWithAnotherConfig 1465  
netErrAuthFailure 1459  
netErrBadScript 1458  
netErrBufTooSmall 1433, 1436, 1450, 1465, 1479  
netErrBufWrongSize 1450, 1457, 1479, 1483  
netErrConfigAliasErr 1433, 1465  
netErrConfigBadName 1434, 1435  
netErrConfigCantDelete 1433, 1434, 1465  
netErrConfigCantPointToAlias 1427  
netErrConfigEmpty 1433, 1465  
netErrConfigNotAlias 1426, 1427  
netErrConfigNotFound 1430, 1433, 1465  
netErrConfigTooMany 1435

netErrDNSAborted 1440, 1442, 1444  
netErrDNSAllocationFailure 1440, 1441, 1443  
netErrDNSBadName 1440, 1441, 1443  
netErrDNSBadProtocol 1440, 1442, 1444  
netErrDNSFormat 1440, 1441, 1443  
netErrDNSImpossible 1440, 1442, 1444  
netErrDNSIrrelevant 1440, 1442, 1444  
netErrDNSLabelTooLong 1440, 1441, 1443  
netErrDNSNameTooLong 1440, 1441, 1443  
netErrDNSNIY 1440, 1442, 1443  
netErrDNSNonexistentName 1440, 1442, 1443  
netErrDNSNoPort 1440, 1442, 1444  
netErrDNSNoRecursion 1440, 1442, 1444  
netErrDNSNoRRS 1440, 1442, 1444  
netErrDNSNotInLocalCache 1440, 1442, 1444  
netErrDNSRefused 1440, 1442, 1443  
netErrDNSServerFailure 1440, 1441, 1443  
netErrDNSTimeout 1440, 1441, 1443  
netErrDNSTruncated 1440, 1442, 1444  
netErrDNSUnreachable 1440, 1441, 1443  
netErrInterfaceDown 1475, 1477  
netErrInterfaceNotFound 1446, 1447, 1448, 1450, 1457, 1458, 1465, 1475, 1477  
netErrInternal 1491, 1492  
netErrInvalidInterface 1449  
netErrInvalidSettingSize 1483, 1496, 1498  
netErrIPCan'tFragment 1475, 1477  
netErrIPktOverflow 1475, 1477  
netErrIPNoDst 1475, 1477  
netErrIPNoRoute 1475, 1477  
netErrIPNoSrc 1475, 1477  
netErrMessageTooBig 1475, 1477  
netErrNoInterfaces 1436, 1463, 1465, 1490, 1492  
netErrNoMoreSockets 1494  
netErrNoMultiPacketAddr 1502  
netErrNoMultiPktAddr 1475, 1477  
netErrNotOpen 1425, 1438, 1440, 1441, 1443, 1445, 1448, 1458, 1460, 1468, 1469, 1472, 1474, 1477, 1485, 1486, 1488, 1489, 1490, 1492, 1493, 1496, 1497, 1501, 1502, 1503  
netErrOutOfCmdBlocks 1426, 1427, 1429, 1430, 1431, 1433, 1434, 1435, 1436, 1465, 1475, 1477, 1488, 1489, 1491, 1492, 1494, 1502  
netErrOutOfMemory 1438, 1463, 1465, 1494

---

netErrOutOfPackets 1475, 1477  
netErrOutOfResources 1492  
netErrParamErr 1426, 1427, 1429, 1433, 1434, 1438, 1460, 1465, 1468, 1469, 1474, 1477, 1485, 1486, 1488, 1489, 1490, 1492, 1494, 1496, 1497, 1502  
netErrPortInUse 1490, 1492  
netErrPPPAddressRefused 1459  
netErrPPPTimeout 1458  
netErrPrefNotFound 1433, 1449, 1450, 1457, 1463, 1465, 1479  
netErrQuietTimeNotElapsed 1490, 1492  
netErrReadOnlySetting 1457, 1483  
netErrSocketAlreadyConnected 1488, 1491, 1492  
netErrSocketBusy 1490, 1492  
netErrSocketClosedByRemote 1475, 1477, 1485, 1487, 1488, 1491, 1492  
netErrSocketInputShutdown 1502  
netErrSocketNotConnected 1475, 1477, 1485  
netErrSocketNotListening 1485  
netErrSocketNotOpen 1438, 1468, 1469, 1474, 1477, 1485, 1487, 1488, 1489, 1490, 1492, 1496, 1497, 1502  
netErrStillOpen 1425  
netErrTimeout 1438, 1439, 1440, 1441, 1443, 1445, 1465, 1468, 1469, 1474, 1477, 1485, 1486, 1488, 1489, 1490, 1492, 1493, 1496, 1497, 1501  
netErrTooManyInterfaces 1446  
netErrTooManyTCPConnections 1491  
netErrUnimplemented 1450, 1457, 1460, 1485, 1496, 1497  
netErrUnknownProtocol 1445  
netErrUnknownService 1445  
netErrUnknownSetting 1450, 1457, 1479, 1483  
netErrUnreachableDest 1475, 1477  
netErrUserCancel 1438, 1458, 1468  
netErrWouldBlock 1438, 1468, 1470, 1475, 1477, 1491  
netErrWrongSocketType 1485, 1491, 1492, 1496, 1498  
netFDIsSet 1472  
netFDSet 1472  
netFDSetSize 1473  
NetFDSetType 1471  
netFDZero 1472  
NetHostInfoBufType 1414  
NetHostInfoType 1414  
NetHToNL 1422  
NetHToNS 1423  
netIFMediaDown 96  
netIFMediaUp 96  
NetIFSettingEnum 1450, 1451, 1457  
netIOFlagDontRoute 1421  
netIOFlagOutOfBand 1421  
netIOFlagPeek 1421  
NetIOParamType 1470  
NetIOVecPtr 1470  
NetIOVecType 1470  
NetIPAddr 1414, 1423, 1424  
NetLibAddrAToIN 1423  
NetLibAddrINToA 1424  
NetLibClose 1424  
NetLibConfigAliasGet 1426  
NetLibConfigAliasSet 1427  
NetLibConfigDelete 1429  
NetLibConfigIndexFromName 1430  
NetLibConfigList 1431  
NetLibConfigMakeActive 1432  
NetLibConfigRename 1434  
NetLibConfigSaveAs 1435  
NetLibConnectionRefresh 1436  
NetLibDmReceive 1437  
NetLibFinishCloseWait 1438  
NetLibGetHostByAddr 1439  
NetLibGetHostByName 1441  
NetLibGetMailExchangeByName 1442  
NetLibGetServByName 1444  
NetLibIFAttach 1446  
NetLibIFDetach 1447  
NetLibIFDown 1448  
NetLibIFGet 1449  
NetLibIFSettingGet 1450  
NetLibIFSettingSet 1457  
NetLibIFUp 1458  
NetLibMaster 1459  
NetLibOpen 1463  
NetLibOpenConfig 1464  
NetLibOpenCount 1466

---

NetLibReceive 1467, 1508  
NetLibReceivePB 1469  
NetLibSelect 1471  
NetLibSend 1474, 1509  
NetLibSendPB 1476  
NetLibSettingGet 1479  
NetLibSettingSet 1483  
NetLibSocketAccept 1484, 1485  
NetLibSocketAddr 1486  
NetLibSocketBind 1487  
NetLibSocketClose 1489  
NetLibSocketConnect 1490, 1508  
NetLibSocketListen 1491, 1492  
NetLibSocketOpen 1493, 1508  
NetLibSocketOptionGet 1495  
NetLibSocketOptionSet 1497  
NetLibSocketShutdown 1501  
NetLibTracePrintF 1502  
NetLibTracePutS 1503  
NetMasterEnum 1459  
netMasterICMPStats command 1462  
netMasterInterfaceInfo command 1460  
netMasterInterfaceStats command 1461  
netMasterIPStats command 1462  
NetMasterPBPtr 1459  
netMasterTCPStats command 1462  
netMasterTraceEventGet command 1462  
netMasterUDPStats command 1462  
NetMgr.h 1413  
NetNToHL 1504  
NetNToHS 1504  
NetServInfoBufType 1416  
NetServInfoType 1416  
NetSettingEnum 1479, 1483  
NetSocket.c 1507  
NetSocketAddrEnum 1417  
netSocketAddrINET 1494  
NetSocketAddrINType 1417  
netSocketAddrRaw 1494  
NetSocketAddrRawType 1418  
NetSocketAddrType 1419  
netSocketDirBoth 1501  
NetSocketDirEnum 1501  
netSocketDirInput 1501  
netSocketDirOutput 1501  
NetSocketLingerType 1499  
NetSocketOptEnum 1495, 1497, 1498  
NetSocketOptLevelEnum 1495, 1497, 1498  
netSocketProtoIPRAW 1493  
netSocketProtoIPTCP 1493  
netSocketProtoIPUDP 1493  
NetSocketRef 1493  
NetSocketTypeEnum 1419  
netTracingAppMsgs 1421  
netTracingAppPktIP 1421  
netTracingData 1422  
netTracingData40 1422  
netTracingErrors 1421  
netTracingFuncs 1421  
netTracingIFHi 1422  
netTracingIFLow 1422  
netTracingIFMid 1422  
netTracingMsgs 1421  
netTracingPkts 1421  
NetUReadN 1507  
NetUTCPOpen 1508  
NetUWriteN 1509  
new serial manager 2320, 2321  
NFNT resource 718  
nfnt resource 721  
nilEvent 62, 959, 960  
noFocus 272, 274, 287, 317  
noListSelection 345  
noMenuItemSelection 366, 370  
noMenuSelection 366, 370  
noPreferenceFound 841, 843  
noteTextTableItem 434  
notification manager 2330  
notifyDetailsP 94  
NotifyMgr.h 71, 801  
NumberFormatType 764, 770  
numericGraffitiSilkscreenArea 954  
numericTableItem 425  
numLockMask 57  
numUneditableCategories 136, 139, 147

---

## O

off-screen windows 1168  
olume Constants 884  
omErrBadOverlayDBName 819  
omErrBaseRequiresOverlay 569  
omErrDatabaseRequiresOverlay 614  
omErrInvalidLocaleIndex 814  
omErrNoNextSystemLocale 815  
omErrUnknownLocale 569, 818, 819  
omFtrCreator 812  
omFtrDefaultLocale 812  
omFtrShowErrorsFlag 812  
OmGetCurrentLocale 813  
OmGetIndexedLocale 814  
OmGetNextSystemLocale 815  
OmGetRoutineAddress 816  
OmGetSystemLocale 773, 817  
OmGlueGetCurrentLocale 813, 1892  
OmGlueGetSystemLocale 817, 1892  
OmLocaleToOverlayDBName 818  
OmLocaleType 764, 811  
OmOverlayDBNameToLocale 819  
omOverlayDBType 613  
omOverlayRscID 812  
omOverlayRscType 812  
OmSelector 816  
OmSetSystemLocale 820  
open count of net library 1466  
opening net library 1463, 1464  
optionKeyMask 57  
OverlayMgr.h 811, 816  
overlays 613

## P

Palm OS 2.0 feature set 2304  
Palm OS 3.0 feature set 2308  
Palm OS 3.1 feature set 2312  
Palm OS 3.2 feature set 2315  
Palm OS 3.5 feature set 2324, 2330, 2342, 2346, 2351  
PalmLocale.h 763, 765, 767  
PalmOSGlue.lib 1891, 2313, 2316  
panel list (*SysCreatePanelList*) 972  
password functions 823

Password.h 823  
PatternType 1153  
PceNativeCall 1258  
PdiDefineReaderDictionary 2253  
PdiDefineResizing 2254  
PdiDefineWriterDictionary 2255  
PdiEnterObject 2256  
PdiLibClose 2257  
PdiLibOpen 2257  
PdiParameterPairTest 2258  
PdiReaderDelete 2259  
PdiReaderNew 2259  
PdiReaderType 2238  
PdiReadParameter 2260  
PdiReadProperty 2261  
PdiReadPropertyField 2262  
PdiReadPropertyName 2264  
PdiSetCharset 2265  
PdiSetEncoding 2266  
PdiWriteBeginObject 2266  
PdiWriteEndObject 2267  
PdiWriteParameter 2268  
PdiWriteParameterStr 2270  
PdiWriteProperty 2271  
PdiWritePropertyValueBinaryValue 2272  
PdiWritePropertyFields 2273  
PdiWritePropertyStr 2274  
PdiWritePropertyValue 2275  
PdiWriterDelete 2276  
PdiWriterNew 2276  
PdiWriterType 2240  
PeerCert SSL Attribute 2202  
PeerCommonName SSL Attribute 2202  
pen  
    current status 950  
pen manager functions 825–826  
pen queue  
    flushing 949  
    size 956  
PenBtnInfoType 951  
PenCalibrate 825  
penDownEvent 44, 47, 49, 53, 57, 62, 66, 171, 221, 299, 347, 382, 412  
PenMgr.h 825

---

penMoveEvent 62  
PenResetCalibration 826  
penUpEvent 63, 945  
PhoneLookup.h 493  
PhoneNumberLookup 498  
PhoneNumberLookupCustom 499  
PilotMain 995  
PixelFormatType 533  
PluginCallbackProcType 1511  
PluginCmdPtr 1512  
PluginCmdType 1512  
PluginExecCmdType 1513, 1515  
PluginInfoPtr 1514  
PluginInfoType 1514, 1515  
pluginMaxNumOfcmds 1514  
pluginNetLibCallUIProc 1516, 1519, 1521  
pluginNetLibCheckCancelStatus 1516, 1520  
pluginNetLibConnLog 1516, 1521  
pluginNetLibDoNothing 1516, 1519  
pluginNetLibGetSerLibRefNum 1517, 1521  
pluginNetLibGetUserName 1516, 1520  
pluginNetLib GetUserPwd 1516, 1520  
pluginNetLibPromptUser 1516, 1520  
pluginNetLibReadBytes 1516, 1519  
pluginNetLibWriteBytes 1516, 1520  
PointType 853  
popSelectEvent 63, 298, 299  
popup list 350  
popupTriggerTableItem 425  
port... constants 1537  
poweredOnKeyMask 57  
prefAlarmSoundVolume 875  
preferenceDataVer2 841  
preferenceDataVer3 841  
preferenceDataVer4 841  
preferenceDataVer5 841  
preferenceDataVer6 841  
preferenceDataVer8 841  
preferenceDataVer9 841  
preferenceDataVerLatest 841, 2314  
preferenceDataVersion 2314  
preferences  
  changing with launch codes 32  
  multiple application preferences 847  
Preferences.h 763, 764, 827  
prefGameSoundVolume 875  
PrefGetAppPreferences 842, 2306  
PrefGetAppPreferencesV10 844  
PrefGetPreference 94, 845  
PrefGetPreferences 845  
PrefOpenPreferenceDB 846  
PrefOpenPreferenceDBV10 847  
PrefSetAppPreferences 847  
PrefSetAppPreferencesV10 849  
PrefSetPreference 850  
PrefSetPreferences 850  
prefShowPrivateRecords 390, 391  
prefSysSoundVolume 875  
prefTimeZone 194  
PrgCallbackData 400  
PrgCallbackFunc 400  
PrgHandleEvent 393  
PrgStartDialog 394  
PrgStartDialogV31 396  
PrgStopDialog 397  
PrgUpdateDialog 398  
PrgUserCancel 399  
printable characters 1009  
printf 903  
PrivateRecords.h 389  
privateRecordViewEnum 389, 835  
procedure alarms 508  
progress manager 2309  
Progress Manager callback function 400  
Progress.h 393  
ProtocolVersion SSL Attribute 2203  
putc 904  
putchar 904  
puts 905  
PwdExists 823  
PwdRemove 823  
PwdSet 824  
PwdVerify 824

**Q**

query callback function 1409

---

## R

RAM-based heaps 791  
RbufSize SSL Attribute 2204  
RctCopyRectangle 854  
RctGetIntersection 854  
RctInsetRectangle 855  
RctOffsetRectangle 856  
RctPtInRectangle 857  
RctSetRectangle 857  
read callback function 1353  
ReadBufPending SSL Attribute 2204  
ReadOutstanding SSL Attribute 2205  
ReadProc 1315  
ReadRecPending SSL Attribute 2205  
ReadStreaming SSL Attribute 2205  
records  
  deleting 584  
  detaching 585  
  ID 587  
  retrieving information 622  
Rect.h 853  
RectanglePtr 855  
rectangles  
  copying 854  
  erasing 1183  
  intersecting 854  
  moving 856  
  resizing 855  
  scrolling 1229  
RectangleType 853, 855  
reinitializing dynamic memory heap 989  
repeat control object  
  and ctlRepeatEvent 44  
repeating button 44  
reset 35, 989  
ResGlueLoadConstant 501, 1892  
ResLoadConstant 501  
ResLoadForm 502  
ResLoadMenu 502  
resource database (SysCurAppDatabase) 973  
resource ID 561  
resource type 562, 589  
resources  
  retrieving 600

  retrieving information 628  
  searching for 629  
resumeSleepChr 99, 100  
RGBColorType 534  
ROM-based heaps 785, 791  
ROM-based records 618, 620

## S

Sampled Sound Application-Defined Functions 896  
Sampled Sound Functions 884  
Sampled Sound Structures, Constants, and Data Types 879  
SclDrawScrollBar 410  
sclEnterEvent 64, 300, 412  
sclExitEvent 65, 412  
SclGetScrollBar 410  
SclHandleEvent 64, 65, 411  
sclRepeatEvent 65, 300, 412  
  and sclExitEvent 65  
SclSetScrollBar 412  
scptLaunchCmdDoNothing 1515  
scptLaunchCmdExecuteCmd 3, 1513, 1515  
scptLaunchCmdListCmds 3, 1514, 1515  
ScrDisplayMode 1228, 2311, 2327  
ScrDisplayModeOperation 1228  
ScreenAttrType 1222  
ScriptPlugin.h 1511  
ScriptPluginLaunchCodesEnum 1515  
ScriptPluginSelectorProcPtr 1518  
scroll arrows  
  FrmUpdateScrollers 323  
scroll position in field 214  
scrollbar functions 410–415  
scrollbar objects  
  fields 407  
  in tables 448  
  structure 407  
ScrollBar.h 405  
ScrollBarAttrType 405  
ScrollBarPtr 406  
ScrollBarRegionType 406  
ScrollBarType 407  
scrolling rectangle in window 1229

---

ScrOperation 1162  
searching for string 252  
secret records, removing 626  
SecSelectViewStatus 390  
Security application 34  
SecurityAutoLockType 827  
SecVerifyPW 391  
SelDay.h 185  
SelectDay 188, 2306  
selectDayByDay 188  
selectDayByMonth 188  
selectDayByWeek 188  
SelectDayV10 189  
selection in field 215  
SelectOneTime 189  
SelectTime 191  
SelectTimeZone 193  
SelTime.h 185  
SelTimeZone.h 185  
separatorItemSelection 370  
SerClearErr 1595, 1598  
SerClose 1596  
SerControl 1597  
serCtlBreakStatus (in SerCtlEnum) 1594  
serCtlEmuSetBlockingHook (in SerCtlEnum) 1594  
SerCtlEnum 1593  
serCtlFirstReserved (in SerCtlEnum) 1593  
serCtlHandshakeThreshold (in SerCtlEnum) 1594  
serCtlLAST (in SerCtlEnum) 1594  
serCtlMaxBaud (in SerCtlEnum) 1594  
serCtlStartBreak (in SerCtlEnum) 1593  
serCtlStartLocalLoopback (in SerCtlEnum) 1594  
serCtlStopBreak (in SerCtlEnum) 1593  
serCtlStopLocalLoopback (in SerCtlEnum) 1594  
serDev... constants 1559  
serErrAlreadyOpen 1596, 1600  
serErrBadParam 1597, 1600, 1609  
serErrBadPort 1585  
serErrConfigurationFailed 1578, 1585  
serErrLineErr 1595, 1596, 1599, 1601, 1602, 1603,  
  1604  
serErrNotOpen 1585, 1586, 1587, 1596, 1597, 1598,  
  1609  
serErrNotSupported 1585, 1586, 1587  
serErrStillOpen 1596  
serErrTimeOut 1585, 1601, 1602, 1604, 1605, 1606,  
  1607  
serFncConsole 1529, 1556  
serFncDebugger 1529, 1556  
serFncHotSync 1529, 1556  
serFncPPPSession 1529, 1556  
serFncSLIPSession 1529, 1556  
serFncTelephony 1529, 1556  
serFncUndefined 1529, 1556  
SerGetSettings 1598  
SerGetStatus 1598  
serial capabilities constants 1559  
Serial Library 1600  
serial port feature constants 1537  
serial settings constants 1560  
serial status constants 1562  
SerialDrv.h 1523  
SerialLinkMgr.h 1611  
SerialMgr.h 1551  
SerialMgrOld.h 1593  
SerialVdrv.h 1523  
serLineError... constants 1599  
SerOpen 1599  
serPortConsolePort 1557  
serPortCradlePort 1557  
serPortCradleRS232Port 1558  
serPortCradleUSBPort 1558  
serPortIrPort 1557  
serPortLocalHotSync 1557  
SerReceive 1601  
SerReceive10 1602  
SerReceiveCheck 1603  
SerReceiveFlush 1603, 1604  
SerReceiveWait 1604  
SerSend 1605  
SerSend10 1606  
SerSendFlush 1607  
SerSendWait 1607  
SerSetReceiveBuffer 1608  
SerSetSettings 1608  
SerSettingsType 1594  
servent 1416

---

SessionReused SSL Attribute 2206  
sethostname 1481  
shiftKeyMask 57  
silk-screen buttons  
    EvtGetPenBtnList 951  
SilkscreenAreaType 953  
silkscreenRectGraffiti 954  
silkscreenRectScreen 954  
SioAddCommand 905  
SioClearScreen 915  
SioExecCommand 915  
Siofgetc 899, 902, 906  
Siofgets 900, 906  
Siofprintf 900, 907  
Siofputc 901, 904, 908  
Siofputs 902, 908  
SioFree 916  
Siogets 903, 909  
SioHandleEvent 916  
SioInit 917  
SioMain 918  
Sioprintf 903, 909  
Sioputs 905, 910  
Siosystem 910, 912  
Siovfprintf 911, 913  
SleepEventParamType 100  
SliderControlType 164  
SlkClose 1611  
SlkCloseSocket 1612  
slkErrAlreadyOpen 1613  
slkErrBadParam 1617  
slkErrBuffer 1615  
slkErrChecksum 1615  
slkErrNotOpen 1611  
slkErrOutOfSockets 1614  
slkErrSocketNotOpen 1612, 1613, 1615, 1616, 1617,  
    1618, 1619  
slkErrTimeOut 1615, 1616  
slkErrWrongDestSocket 1615  
SlkFlushSocket 1612  
SlkOpen 1613  
SlkOpenSocket 1614  
SlkPktHeaderType 1618  
SlkReceivePacket 1615  
SlkSendPacket 1616  
SlkSetSocketListener 1617  
SlkSocketListenType 1617, 1618  
SlkSocketPortID 1618  
SlkSocketSetTimeout 1619  
SlkWriteDataType 1617  
slLib 2164  
SmsLib.h 2219  
SmsParamsType 2219  
SmsPrefType 2222  
SmsReceiveCDMAParamsType 2223  
SmsReceiveGSMParamsType 2225  
SmsReceiveParamsType 2225  
SmsReceiveTDMAParamsType 2227  
SMSReportParamsType 2227  
SmsSendCDMAParamsType 2228  
SMSendParamsType 2230  
SmsSendTDMAParamsType 2232  
sndAlarm 876  
sndAlarmVolume 884  
SndBlockingFuncType 877  
SndCallbackInfoType 860  
sndClick 876  
sndCmdFreqDurationAmp 870  
sndCmdFrqOn 870  
SndCmdIDType 861  
sndCmdNoteOn 871  
sndCmdQuiet 871  
SndCommandType 861  
SndComplFuncType 878  
sndConfirmation 876  
SndCreateMidiList 868  
SndDoCmd 869  
sndError 876  
sndFlagAsync 883  
sndFlagNormal 883  
sndFlagSync 883  
sndFloat 880  
sndFloatBig 880  
sndFloatLittle 880  
sndFloatOpposite 880  
sndGameVolume 884  
SndGetDefaultVolume 871

sndInfo 876  
sndInput 882  
sndInt16 880  
sndInt16Big 880  
sndInt16Little 880  
sndInt16Opposite 880  
sndInt32 880  
sndInt32Big 880  
sndInt32Little 880  
sndInt32Opposite 880  
sndInt8 880  
SndInterruptSmfIrregardless 872  
sndMaxAmp 833, 862  
SndMidiListItemType 862  
sndMidiNameLength 863  
SndMidiRecHdrType 863  
sndMidiRecSignature 864  
sndMono 883  
sndOutput 882  
sndPanCenter 883  
sndPanFullLeft 883  
sndPanFullRight 883  
SndPlayResource 884  
SndPlaySMF 872  
SndPlaySmfIrregardless 874  
SndPlaySmfResource 875  
SndPlaySmfResourceIrregardless 876  
SndPlaySystemSound 876  
SndPtr 879  
SndSampleType 879  
SndSampleTypeTag 880  
SndSetDefaultVolume 877  
SndSmfCallbacksType 864  
SndSmfChanRangeType 865  
SndSmfCmdEnum 865  
SndSmfOptionsType 865  
sndSmfPlayAllMilliSec 867  
sndStartUp 876  
sndStereo 883  
SndStreamBufferCallback 896  
SndStreamDelete 889  
SndStreamGetPan 890  
SndStreamGetVolume 890  
SndStreamMode 881  
SndStreamModeTag 881  
SndStreamPause 891  
SndStreamRef 882  
SndStreamSetVolume 893  
SndStreamStart 894  
SndStreamStop 895  
SndStreamWidth 882  
SndStreamWidthTag 882  
SndSysBeepType 867  
sndSystemVolume 884  
sndUInt8 880  
sndWarning 876  
sockaddr 1419  
sockaddr\_in 1417  
socket 1508  
socket listener 1616  
socket listener procedure 1616, 1617, 1618  
Socket SSL Attribute 2207  
soft reset 35, 989  
sorting array elements 983  
sorting text 559  
SortRecordInfoType 562  
Sound Manager 859  
sound manager 2309  
Sound Resource Playback Flags 883  
SoundMgr.h 859  
sprintf 912  
sprintf (StrPrintF) 934  
SrmClearErr 1563  
SrmClose 1563  
SrmControl 1552, 1558, 1560, 1564, 1569  
srmControlCustom 1567  
SrmCtlEnum 1552, 1564  
SrmCustomControl 1567  
srmDefaultCTSTimeout 1569  
SrmExtOpen 1557, 1568, 1570, 1575  
SrmExtOpenBackground 1557, 1570, 1576  
SrmGetDeviceCount 1571  
SrmGetDeviceInfo 1551, 1572  
SrmGetStatus 1562, 1573, 1578, 1579, 1581  
SrmOpen 1557, 1570, 1574  
SrmOpenBackground 1557, 1575

---

SrmOpenConfigType 1555, 1568, 1570  
SrmPrimeWakeupHandler 1576, 1590  
SrmRcvQType 1527  
SrmReceive 1577, 1582, 1584, 1590  
SrmReceiveCheck 1579, 1582  
SrmReceiveFlush 1579  
SrmReceiveWait 1581, 1584  
SrmReceiveWindowClose 1582, 1591  
SrmReceiveWindowOpen 1583, 1591  
SrmSend 1584, 1587  
SrmSendCheck 1585  
SrmSendFlush 1586  
SrmSendWait 1587  
SrmSetReceiveBuffer 1569, 1581, 1588  
srmSettings... constants 1560  
SrmSetWakeupHandler 1577, 1589, 1590  
srmStatus... constants 1562  
SSL Alerts 2214  
SSL Attribute Data Types 2182  
SSL Attributes 2187  
SSL Attributes and Macros 2181  
SSL Certificate Errors 2216  
SSL Cryptography Errors 2215  
SSL Data Types 2163  
SSL Function Protocol Errors 2213  
SSL Handshake Errors 2215  
SSL Illegal Message Errors 2216  
SSL Macro Names 2181  
SSL Structures 2166  
SSL Structures and Data Types 2163  
SslAttribute 2163  
SslCallback 2166  
SslCipherSuiteInfo 2167  
SslContext 2163  
SslContextGet\_AppInt32 2187  
SslContextGet\_AppPtr 2188  
SslContextGet\_AutoFlush 2188  
SslContextGet\_BufferedReuse 2189  
SslContextGet\_CipherSuite 2190  
SslContextGet\_CipherSuiteInfo 2190  
SslContextGet\_CipherSuites 2191  
SslContextGet\_Compat 2192  
SslContextGet\_DontSendShutdown 2193  
SslContextGet\_DontWaitForShutdown 2193  
SslContextGet\_Error 2194  
SslContextGet\_HsState 2194  
SslContextGet\_InfoCallback 2195  
SslContextGet\_IoFlags 2197  
SslContextGet\_IoStruct 2198  
SslContextGet\_IoTimeout 2197  
SslContextGet\_LastAlert 2198  
SslContextGet\_LastApi 2199  
SslContextGet\_LastIO 2200  
SslContextGet\_Mode 2201  
SslContextGet\_PeerCert 2202  
SslContextGet\_ProtocolVersio 2203  
SslContextGet\_RbufSize 2204  
SslContextGet\_ReadBufPending 2204  
SslContextGet\_ReadOutstanding 2205  
SslContextGet\_ReadRecPending 2205  
SslContextGet\_ReadStreaming 2206  
SslContextGet\_SessionReused 2206  
SslContextGet\_Socket 2207  
SslContextGet\_SslSession 2208  
SslContextGet\_SslVerify 2208  
SslContextGet\_Streaming 2209  
SslContextGet\_VerifyCallback 2209  
SslContextGet\_WbufSize 2210  
SslContextGet\_WriteBufPending 2210  
SslContextSet\_AppInt32 2187  
SslContextSet\_AppPtr 2188  
SslContextSet\_AutoFlush 2188  
SslContextSet\_CipherSuites 2191  
SslContextSet\_Compat 2192  
SslContextSet\_DontSendShutdown 2193  
SslContextSet\_DontWaitForShutdown 2193  
SslContextSet\_Error 2194  
SslContextSet\_InfoCallback 2195  
SslContextSet\_IoFlags 2197  
SslContextSet\_IoStruct 2198  
SslContextSet\_IoTimeout 2197  
SslContextSet\_Mode 2201  
SslContextSet\_ProtocolVersion 2203  
SslContextSet\_RbufSize 2204  
SslContextSet\_ReadStreaming 2206  
SslContextSet\_Socket 2207

---

SslContextSet\_VerifyCallback 2209  
SslContextSet\_WbufSize 2210  
sslErrBadArgument 2213  
sslErrBadDecode 2216  
sslErrBadLength 2214  
sslErrBadOption 2214  
sslErrBadPeerFinished 2215  
sslErrBadSignature 2216  
sslErrBufferTooSmall 2214  
sslErrCbAbort 2214  
sslErrCert 2216  
sslErrCertDecodeError 2216  
sslErrCsp 2215  
sslErrDivByZero 2215  
sslErrEof 2214  
sslErrExtraHandshakeData 2215  
sslErrFailed 2214  
sslErrFatalAlert 2214  
sslErrHandshakeEncoding 2215  
sslErrHandshakeProtocol 2215  
sslErrIo 2214  
sslErrNoModInverse 2215  
sslErrNoRandom 2215  
sslErrNullArg 2214  
sslErrOutOfMemory 2214  
sslErrReadAppData 2215  
sslErrRecordError 2216  
sslErrUnexpectedRecord 2216  
sslErrUnsupportedCertType 2216  
sslErrUnsupportedSignatureType 2216  
sslErrVerifyBadSignature 2216  
sslErrVerifyConstraintViolation 2216  
sslErrVerifyNotAfter 2216  
sslErrVerifyNotBefore 2217  
sslErrVerifyNoTrustedRoot 2217  
sslErrVerifyUnknownCriticalExtension 2217  
sslErrWrongMessage 2216  
SslExtendedItem 2169  
SslExtendedItems 2171  
SslIoBuf 2172  
SslLibCallback 2166, 2173, 2174  
SslLibGet\_AppInt32 2187  
SslLibGet\_AppPtr 2188  
SslLibGet\_AutoFlush 2188  
SslLibGet\_CipherSuites 2191  
SslLibGet\_Compat 2192  
SslLibGet\_DontSendShutdown 2193  
SslLibGet\_DontWaitForShutdown 2193  
SslLibGet\_InfoCallback 2195  
SslLibGet\_Mode 2201  
SslLibGet\_ProtocolVersion 2203  
SslLibGet\_RbufSize 2204  
SslLibGet\_ReadStreaming 2206  
SslLibGet\_VerifyCallback 2209  
SslLibGet\_WbufSize 2210  
SslLibSet\_AppInt32 2187  
SslLibSet\_AppPtr 2188  
SslLibSet\_AutoFlush 2188  
SslLibSet\_CipherSuites 2191  
SslLibSet\_Compat 2192  
SslLibSet\_DontSendShutdown 2193  
SslLibSet\_DontWaitForShutdown 2193  
SslLibSet\_InfoCallback 2195  
SslLibSet\_Mode 2201  
SslLibSet\_ProtocolVersion 2203  
SslLibSet\_RbufSize 2204  
SslLibSet\_ReadStreaming 2206  
SslLibSet\_VerifyCallback 2209  
SslLibSet\_WbufSize 2210  
SslSession 2174  
SslSession SSL Attribute 2207  
SslSocket 2176  
SslVerify 2177  
SslVerify SSL Attribute 2208  
standard IO functions 899  
StartApplication  
    and PrefGetPreferences 846  
stat 690  
stdFont 711, 736, 1903  
StdIOPalm.h 899  
StdIOProvider.h 899  
Stereo Pan Constants 883  
StrAToI 919  
StrCaselessCompare 920  
StrCat 921  
strcat function substitute (StrCat) 921

---

StrChr 921  
strchr function substitute (StrChr) 921  
strcmp function substitute (StrCompare) 922  
StrCompare 922  
StrCompareAscii 923  
StrCopy 924  
strcpy function substitute (StrCopy) 924  
StrDelocalizeNumber 924  
StrDelocalizeNumber, and launch code 16  
Streaming SSL Attribute 2209  
stricmp function substitute  
  (StrCaselessCompare) 920  
string manager 919–940  
string resource  
  copying 969  
string searching 252  
StringMgr.h 919  
StrIToA 925  
StrIToH 926  
StrLen 926  
strlen function substitute (StrLen) 926  
StrLocalizeNumber 927  
  launch code 16  
StrNCaselessCompare 929  
StrNCat 930  
strncat function substitute (StrNCat) 930  
strncmp 932  
StrNCompare 931  
StrNCompareAscii 932  
StrNCopy 933  
strokes, translating 956  
StrPrintF 912, 934  
StrStr 935  
strstr function substitute (StrStr) 935  
StrToLower 935  
structure of field object 198  
StrVPrintF 914, 936  
summary of launch codes 3, 71, 749  
symbol11Font 712  
symbol7Font 712  
symbolFont 711  
sys\_socket.h 1507  
SysAlarmTriggeredParamType 9, 510  
SysAppLaunch 962

sysAppLaunchCmdAddRecord 6  
sysAppLaunchCmdAlarmTriggered 9, 17, 507  
sysAppLaunchCmdCardLaunch 15  
SysAppLaunchCmdCardType 16  
sysAppLaunchCmdCountryChange 16  
sysAppLaunchCmdDisplayAlarm 9, 16, 507  
sysAppLaunchCmdExgAskUser 17, 19, 1297,  
  1322, 1335, 1337, 2308  
sysAppLaunchCmdExgGetData 19  
sysAppLaunchCmdExgPreview 19, 22, 1310, 1323,  
  1336, 1342  
sysAppLaunchCmdExgReceiveData 21, 22, 1310,  
  1317, 1320, 1323, 1342, 2308  
sysAppLaunchCmdFind 22, 1918  
sysAppLaunchCmdGoto 23, 26, 51, 1298, 1320,  
  1334, 2308  
sysAppLaunchCmdGoToURL 27, 1349  
sysAppLaunchCmdHandleSyncCallApp 28  
sysAppLaunchCmdHandleSyncCallAppType 29  
sysAppLaunchCmdInitDatabase 30  
sysAppLaunchCmdLookup 31, 2304  
sysAppLaunchCmdNotify 31, 74, 807  
sysAppLaunchCmdOpenDB 32  
sysAppLaunchCmdPanelCalledFromApp 32, 33,  
  2304  
sysAppLaunchCmdReturnFromPanel 32, 33, 2304  
sysAppLaunchCmdSaveData 23, 33  
sysAppLaunchCmdSyncNotify 34, 1344  
sysAppLaunchCmdSystemLock 34, 2304  
sysAppLaunchCmdSystemReset 35, 97  
sysAppLaunchCmdTimeChange 35  
sysAppLaunchCmdURLParams 36  
SysAppLauncherDialog 417  
sysAppLaunchFlagNewGlobals 26  
sysAppLaunchFlagNewGlobals launch flag 37  
sysAppLaunchFlagSubCal launch flag 37  
sysAppLaunchFlagUIApp launch flag 37  
SysBatteryInfo 963, 2311  
SysBatteryInfoV20 964  
SysBatteryKind 831  
SysBinarySearch 966  
SysBroadcastActionCode 968  
SysCopyStringResource 969  
SysCreateDataBaseList 970

---

SysCreatePanelList 972  
SysCurAppDatabase 973  
SysDBListItemType 970  
SysDisplayAlarmParamType 17  
sysErrLibNotFound 985, 986  
sysErrNoFreeLibSlots 986  
sysErrNoFreeRAM 986  
sysErrOutOfOwnerID 962  
sysErrOutOfOwnerIDs 968  
sysErrParamErr 962, 968, 987  
SysErrString 973  
SysEvent.h 39, 2328  
SysEvtMgr.h 941  
sysExternalConnectorAttachEvent 77  
sysExternalConnectorDetachEvent 77  
SysFatalAlert 418  
sysFileCBtConnectPanelHelper 1559  
sysFileCSmsLib 2220  
sysFileCUart328 1558  
sysFileCUart328EZ 1558  
sysFileCUart650 1558  
sysFileCVirtIrComm 1559  
sysFileCVirtRfComm 1559  
sysFileDescStdIn 1472  
SysFormPointerArrayToStrings 974, 1329, 1330  
sysFtrNewSerialPresent 2321  
sysFtrNewSerialVersion 2322  
sysFtrNumEncoding 2318  
sysFtrNumIntlMgr 2316  
sysFtrNumNotifyMgrVersion 802, 2330  
sysFtrNumProcessor328 2314  
sysFtrNumProcessorEZ 2314  
sysFtrNumProcessorID 2314  
sysFtrNumProcessorIs68K 975  
sysFtrNumProcessorIsARM 976  
sysFtrNumProcessorMask 2314  
sysFtrNumROMVersion 2304, 2308, 2312  
SysGetOSVersionString 977, 2311  
SysGetROMToken 977  
SysGetRomToken 2311  
SysGetStackInfo 978, 2311  
SysGetTrapAddress 979  
SysGlueGetTrapAddress 979, 1892  
SysGlueTrapExists 1912  
SysGraffitiReferenceDialog 418  
SysGremlins 980, 2311  
SysHandleEvent 78, 89, 93, 98, 99, 808, 981  
SysInsertionSort 981  
SysKeyboardDialog 984, 2306  
SysKeyboardDialogV10 984  
SysLibFind 985  
SysLibInstall 985  
SysLibLoad 986  
SysLibRemove 987  
sysNotifyAntennaRaisedEvent 78  
SysNotifyBroadcast 802  
SysNotifyBroadcastDeferred 362, 804  
sysNotifyBroadcasterCode 75, 801  
SysNotifyBroadcastFromInterrupt 805  
sysNotifyCardInsertedEvent 79  
sysNotifyCardRemovedEvent 80  
sysNotifyDBChangedEvent 82  
SysNotifyDBChangedType 83  
sysNotifyDBCreatedEvent 81  
SysNotifyDBCreatedType 81  
sysNotifyDBDeletedEvent 85, 583  
SysNotifyDBDeletedType 85  
sysNotifyDBDirtyEvent 86  
SysNotifyDBDirtyType 87  
SysNotifyDBInfoType 88  
sysNotifyDefaultQueueSize 801, 803  
sysNotifyDeleteProtectedEvent 87  
sysNotifyDeviceUnlocked 88, 93  
SysNotifyDisplayChangeDetailsType 89  
sysNotifyDisplayChangeEvent 88, 1213  
sysNotifyEarlyWakeUpEvent 89  
sysNotifyErrBroadcastBusy 802  
sysNotifyErrDuplicateEntry 807  
sysNotifyErrEntryNotFound 809  
sysNotifyErrNoStackSpace 802  
sysNotifyErrQueueFull 804, 805  
sysNotifyForgotPasswordEvent 90  
sysNotifyGotUsersAttention 91  
sysNotifyHelperEvent 91, 749, 751  
sysNotifyIrDASniffEvent 73  
sysNotifyLateWakeUpEvent 88, 93

---

sysNotifyLocaleChangedEvent 94  
SysNotifyLocaleChangedType 94  
sysNotifyMenuBarOpenEvent 59, 95, 299, 373, 377  
sysNotifyNetLibIFMediaEvent 95  
SysNotifyNetLibIFMediaType 96  
sysNotifyNoDatabaseID 801  
sysNotifyNormalPriority 802, 807  
SysNotifyParamType 32, 74, 78, 751, 807  
sysNotifyPhoneEvent 73  
sysNotifyPOSEMountEvent 73  
SysNotifyProcPtr 806, 808, 810  
SysNotifyRegister 806  
sysNotifyResetFinishedEvent 97  
sysNotifyRetryEnqueueKey 97  
sysNotifySleepNotifyEvent 98, 99  
sysNotifySleepRequestEvent 98, 99  
sysNotifySyncFinishEvent 34, 101  
sysNotifySyncStartEvent 34, 101  
sysNotifyTimeChangeEvent 35, 101, 1069  
SysNotifyUnregister 809  
sysNotifyVersionNum 802  
sysNotifyVolumeMountedEvent 102  
sysNotifyVolumeUnmountedEvent 103  
SysQSort 987  
SysRandom 989  
sysRandomMax 989  
SysReset 989  
sysResIDEExtPrefs 614  
sysResTExtPrefs 614  
SysSetAutoOffTime 990  
SysSetTrapAddress 990  
sysSleepAutoOff 100  
sysSleepPowerButton 100  
sysSleepResumed 100  
sysSleepUnknown 100  
SysStringByIndex 991  
SysTaskDelay 992  
system 912  
system events  
  checking availability 959  
system keyboard display 984  
SystemMgr.h 3, 961  
SystemPreferencesChoice 829, 837

SystemPreferencesType 829  
SystemResources.h 1551  
SysTicksPerSecond 992  
sysTrap.... 979, 990  
SysTraps.h 979, 990  
SysUIAppSwitch 993  
SysUtils.h 961

## T

table functions 433–475  
table objects  
  fields 430  
  structure 430  
Table.h 419  
TableAttrType 419  
TableColumnAttrType 420  
TableDrawItemFuncPtr 433, 451  
TableDrawItemFuncType 475  
TableItemPtr 422  
TableItemStyleType 424  
TableItemType 422  
TableLoadDataFuncType 433, 451, 476  
tableMaxTextItemSize 424  
TablePtr 427  
TableRowAttrType 428  
tables  
  setting load data callback 467  
  setting save data callback 473  
TableSaveDataFuncType 478  
TableType 430  
taIB 535  
taif 535  
TblDrawTable 433  
TblEditing 434  
tblEnterEvent 66, 67, 300  
TblEraseTable 435  
tblExitEvent 67, 68  
TblFindRowData 435  
TblFindRowID 436  
TblGetBounds 436  
TblGetColumnSpacing 437  
TblGetColumnWidth 437  
TblGetCurrentField 438, 446

---

TblGetItemBounds 438  
TblGetItemFont 439, 2311  
TblGetItemInt 440  
TblGetItemPtr 440  
TblGetLastUsableRow 441  
TblGetNumberOfRows 442  
TblGetRowData 443  
TblGetRowHeight 443  
TblGetRowID 444  
TblGetSelection 444  
TblGlueGetColumnMasked 1912  
TblGlueGetNumberOfColumns 442, 1893  
TblGlueGetTopRow 445, 1893  
TblGlueSetSelection 474, 1893  
TblGrabFocus 445  
TblHandleEvent 66, 67, 447  
TblHasScrollBar 448  
TblInsertRow 449  
TblMarkRowInvalid 450  
TblMarkTableInvalid 450  
TblRedrawTable 451  
TblReleaseFocus 452  
TblRemoveRow 453  
TblRowInvalid 453  
TblRowMasked 454  
TblRowSelectable 455  
TblRowUsable 455  
tblSelectEvent 67, 448  
TblSelectItem 456  
TblSetBounds 457  
TblSetColumnEditIndicator 457  
TblSetColumnMasked 458  
TblSetColumnSpacing 459  
TblSetColumnUsable 460  
TblSetColumnWidth 460  
TblSetCustomDrawProcedure 461  
TblsetItemFont 462, 2311  
TblsetItemInt 463  
TblsetItemPtr 464  
TblsetItemStyle 465  
TblSetLoadDataProcedure 467  
TblSetRowData 467  
TblSetRowHeight 468  
TblSetRowID 469  
TblSetRowMasked 470  
TblSetRowSelectable 471  
TblSetRowStaticHeight 472  
TblSetRowUsable 472  
TblSetSaveDataProcedure 473  
TblUnhighlightSelection 475  
tblUnusableRow 441  
tbf 535  
Tbfp 535  
TelCallStateType 1625  
TelCancel 1638  
TelCfgGetPhoneNumber 1685  
TelCfgGetPhoneNumberType 1681  
TelCfgGetSmsCenterType 1682  
TelCfgSetSmsCenter 1688  
TelCgfGetSmsCenter 1687  
TelClose 1639  
TelClosePhoneConnection 1640  
TelDataCallNumberType 1715  
TelDtcCallNumber 1720  
TelDtcCloseLine 1722  
TelDtcReceiveData 1723  
TelDtcReceiveDataType 1716  
TelDtcSendData 1724  
TelDtcSendDataType 1717  
TelEmcCall 1726  
TelEmcCloseLine 1728  
TelEmcGetNumber 1729  
TelEmcGetNumberType 1717  
TelEmcSelectNumber 1732  
TelEmcSetNumber 1734  
TelEmcSetNumberType 1718  
TelephonyMgr.h 2224, 2228, 2229  
TelEventType 1623  
TelGetCallState 1641  
TelGetEvent 1643  
TelGetNumberCount 1731  
TelGetTelephonyEvent 1644  
TelInfGetInformation 1645  
TelInformationType 1627  
TelIsCfgServiceAvailable 1647  
TelIsDtcServiceAvailable 1648

---

TellIsEmcServiceAvailable 1649  
TellIsInfServiceAvailable 1650  
TellIsNwkServiceAvailable 1652  
TellIsOemServiceAvailable 1653  
TellIsPhbServiceAvailable 1654  
TellIsPhoneConnected 1655  
TellIsPowServiceAvailable 1656  
TellIsSmsServiceAvailable 1658  
TellIsSndServiceAvailable 1659  
TellIsSpcServiceAvailable 1660  
TellIsStyServiceAvailable 1661  
TelMatchPhoneDriver 1663  
TelNwkGetLocation 1699  
TelNwkGetLocationType 1695  
TelNwkGetNetworkName 1701  
TelNwkGetNetworkNameType 1696  
TelNwkGetNetworks 1703  
TelNwkGetNetworksType 1697  
TelNwkGetNetworkType 1705  
TelNwkGetSearchMode 1706  
TelNwkGetSelectedNetwork 1269, 1707  
TelNwkGetSignalLevel 1709  
TelNwkSelectNetwork 1711  
TelNwkSetSearchMode 1712  
TelOemCall 1664  
TelOemCallType 1628  
TelOpen 1665  
TelOpenPhoneConnection 1666  
TelPhbAddEntry 1821  
TelPhbDeleteEntry 1823  
TelPhbEntryType 1815  
TelPhbGetAvailablePhonebooks 1824  
TelPhbGetAvailablePhonebooksType 1817  
TelPhbGetEntries 1826  
TelPhbGetEntriesType 1818  
TelPhbGetEntry 1828  
TelPhbGetEntryCount 1830  
TelPhbGetEntryCountType 1819  
TelPhbGetEntryMaxSizes 1832  
TelPhbGetEntryMaxSizesType 1819  
TelPhbGetSelectedPhonebook 1833  
TelPhbSelectPhonebook 1835  
TelPowGetBatteryStatus 1667

TelPowGetPowerLevel 1669  
TelPowSetPhonePower 1670  
TelSendCommandString 1671  
TelSendCommandStringType 1628  
TelSmsDateTimeType 1755  
TelSmsDeleteMessage 1785  
TelSmsDeleteMessageType 1756  
TelSmsDeliveryAdvancedCDMAType 1757  
TelSmsDeliveryAdvancedGSMTYPE 1758  
TelSmsDeliveryAdvancedTDMAType 1759  
TelSmsDeliveryMessageType 1761  
TelSmsExtensionType 1764, 1768, 1769, 1780  
TelSmsGetAvailableStorage 1786  
TelSmsGetAvailableStorageType 1765  
TelSmsGetDataMaxSize 1788  
TelSmsGetMessageCount 1790  
TelSmsGetMessageCountType 1766  
TelSmsGetSelectedStorage 1792  
TelSmsGetUniquePartId 1793  
TelSmsManualAckType 1767  
TelSmsReadMessage 1795  
TelSmsReadMessages 1798  
TelSmsReadMessagesType 1769  
TelSmsReadReport 1800  
TelSmsReadReports 1802  
TelSmsReadReportsType 1770  
TelSmsReadSubmittedMessage 1804  
TelSmsReadSubmittedMessages 1806  
TelSmsReadSubmittedMessagesType 1771  
TelSmsReportType 1772  
TelSmsSelectStorage 1808  
TelSmsSendManualAcknowledge 1809  
TelSmsSendMessage 1811  
TelSmsSendMessageType 1774  
TelSmsSubmitAdvancedCDMAType 1774  
TelSmsSubmitAdvancedGSMTYPE 1775  
TelSmsSubmitAdvancedTDMAType 1777  
TelSmsSubmitMessageType 1778  
TelSmsSubmittedMessageType 1780  
TelSndMute 1672  
TelSndPlayKeyTone 1673  
TelSndPlayKeyToneType 1629  
TelSndStopKeyTone 1675

---

TelSpcAcceptCall 1735  
TelSpcCallNumber 1737  
TelSpcCloseLine 1739  
TelSpcConference 1741  
TelSpcGetCallerNumber 1742  
TelSpcGetCallerNumberType 1718  
TelSpcHoldLine 1744  
TelSpcPlayDTMF 1745  
TelSpcPlayDTMFType 1719  
TelSpcRejectCall 1747  
TelSpcRetrieveHeldLine 1748  
TelSpcSelectLine 1749  
TelSpcSendBurstDTMF 1750  
TelSpcStartContinuousDTMF 1752  
TelSpcStopContinuousDTMF 1753  
TelStyChangeAuthenticationCode 1690  
TelStyChangeAuthenticationType 1683  
TelStyEnterAuthenticationCode 1691  
TelStyGetAuthenticationState 1693  
text clipboard 206  
text manager 997–1040, 2317  
text, finding with GetCharCaselessValue 558  
TextMgr.h 997  
textTableItem 426, 433, 434, 451  
textWithNoteTableItem 426, 433, 434, 451  
TimAdjust 1066  
TimDateTimeToSeconds 1067  
time manager  
  structures 1045  
time system resource 185  
time, displaying and selecting 191  
TimePtr 1054  
timeTableItem 427  
TimeToAscii 1070  
TimeType 1054  
timeZoneStringLength 1071  
TimGetSeconds 1067  
TimGetTicks 1068  
TimSecondsToDateTIme 1068  
TimSetSeconds 101, 1069  
TimTimeZoneToUTC 1072  
TimUTCToTimeZone 1073  
tint 501

titles  
  active area 53  
  copying form title 277  
transliteration 1036  
translitOpLowerCase 1037  
translitOpPreprocess 1037  
TranslitOpType 1037  
translitOpUpperCase 1037  
TsmGlueGetFepMode 1892  
TsmGlueSetFepMode 1892  
TxtByteAttr 998  
TxtCaselessCompare 999  
  and StrCaselessCompare 920, 922, 929, 931  
TxtCharAttr 1000  
TxtCharBounds 1001  
TxtCharEncoding 1003  
TxtCharIsAlNum 1004  
TxtCharIsAlpha 1004  
TxtCharIsCntrl 1005  
TxtCharIsDelim 1005  
TxtCharIsDigit 1006  
TxtCharIsGraph 1006  
TxtCharIsHardKey 1007  
TxtCharIsHex 1008  
TxtCharIsLower 1008  
TxtCharIsPrint 1009  
TxtCharIsPunct 1010  
TxtCharIsSpace 1010  
TxtCharIsUpper 1011  
TxtCharIsValid 1011  
TxtCharSize 1012  
TxtCharWidth 1013  
TxtCharXAttr 1013  
TxtCompare 1014  
TxtConvertEncoding 1017  
TxtEncodingName 1021  
txtErrConvertOverflow 1018  
txtErrTranslitOverflow 1037  
txtErrTranslitOverrun 1037  
txtErrTranslitUnderflow 1037  
txtErrUnknownEncoding 1019  
txtErrUnknownTranslitOp 1037  
TxtFindString 24, 1022, 1914  
  and FindStrInStr 252

---

`TxtGetChar` 1023  
`TxtGetNextChar` 1024, 1030  
`TxtGetPreviousChar` 1025  
`TxtGetTruncationOffset` 1027  
`TxtGetWordWrapOffset` 1027  
`TxtGlueByteAttr` 998, 1893  
`TxtGlueCaselessCompare` 1000, 1893  
`TxtGlueCharAttr` 1001, 1893  
`TxtGlueCharBounds` 1002, 1893  
`TxtGlueCharEncoding` 1004, 1893  
`TxtGlueCharIsAlNum` 1004, 1893  
`TxtGlueCharIsAlpha` 1005, 1893  
`TxtGlueCharIsCntrl` 1005, 1893  
`TxtGlueCharIsDelim` 1006, 1893  
`TxtGlueCharIsDigit` 1006, 1893  
`TxtGlueCharIsGraph` 1007, 1893  
`TxtGlueCharIsHex` 1008, 1893  
`TxtGlueCharIsLower` 1008, 1893  
`TxtGlueCharIsPrint` 1009, 1893  
`TxtGlueCharIsPunct` 1010, 1893  
`TxtGlueCharIsSpace` 1010, 1893  
`TxtGlueCharIsUpper` 1011, 1893  
`TxtGlueCharIsValid` 1012, 1893  
`TxtGlueCharIsVirtual` 1913  
`TxtGlueCharSize` 1012, 1893  
`TxtGlueCharWidth` 1893  
`TxtGlueCharXAttr` 1014, 1894  
`TxtGlueCompare` 1016, 1894  
`TxtGlueEncodingName` 1021, 1894  
`TxtGlueFindString` 24, 252  
`TxtGlueGetChar` 1024, 1894  
`TxtGlueGetHorizEllipsisChar` 1915, 2313  
`TxtGlueGetNextChar` 1025, 1894  
`TxtGlueGetNumericSpaceChar` 1916, 2313  
`TxtGlueGetPreviousChar` 1026, 1894  
`TxtGlueGetTruncationOffset` 1027, 1894  
`TxtGlueLowerChar` 1916  
`TxtGlueLowerStr` 1917  
`TxtGlueMaxEncoding` 1029, 1894  
`TxtGlueNextCharSize` 1031, 1894  
`TxtGlueParamString` 1032, 1894  
`TxtGluePrepFindString` 1022, 1914, 1918  
`TxtGluePreviousCharSize` 1033, 1894

`TxtGlueReplaceStr` 1034, 1894  
`TxtGlueSetNextChar` 1035, 1894  
`TxtGlueStrEncoding` 1036, 1894  
`TxtGlueStripSpaces` 1919  
`TxtGlueTransliterate` 1038, 1894  
`TxtGlueTruncateString` 1920  
`TxtGlueUpperChar` 1921  
`TxtGlueUpperStr` 1922  
`TxtGlueWordBounds` 1039, 1894  
`TxtMaxEncoding` 1028  
`TxtNameToEncoding` 1029  
`TxtNextCharSize` 1030  
`TxtParamString` 1031  
`TxtPreviousCharSize` 1032  
`TxtReplaceStr` 1033  
`TxtSetNextChar` 1034  
`TxtStrEncoding` 1035  
`TxtTransliterate` 1036  
`TxtWordBounds` 1039

## U

`UDABufferSize` 2279  
`UDAControl` 2284  
`UDADelete` 2286  
`UDAEndOfReader` 2286  
`UDAExchangeReaderNew` 2291  
`UDAExchangeWriterNew` 2291  
`UDAFILTERJoin` 2287  
`UDAFILTERType` 2280  
`UDAInitiateWrite` 2287  
`UDAMemoryReaderNew` 2284, 2292  
`UDAMoreData` 2288  
`UDAObjectType` 2280  
`UDARead` 2288  
`UDAReaderType` 2281  
`UDAWriterFlush` 2289  
`UDAWriterJoin` 2290  
`UDAWriterType` 2282  
`UIBrightnessAdjust` 489  
`UIColor.h` 479  
`UIColorGetTableEntryIndex` 483  
`UIColorGetTableEntryRGB` 485  
`UIColorSetTableEntry` 487

- 
- UIColorTableEntries 479  
UICommon.h 493  
UIContrastAdjust 490  
UIControls.h 489  
UIPickColor 490  
UIPickColorStartPalette 491  
UIPickColorStartRGB 491  
UIPickColorStartType 490  
UIResources.h 374  
UIResources.r 614  
UnderlineModeType 197, 1154  
unitsEnglish 827  
unitsMetric 827  
user name  
    obtaining 1251
- V**
- valid characters 1011  
vchrCommand 372, 381, 383, 384, 388  
vchrHardAntenna 2320  
vchrMenu 299, 372, 381, 383, 384, 388, 2328  
vchrRadioCoverageFail 2320  
vchrRadioCoverageOK 2320  
VdrvAPIType structure 1527  
VDrvClose 1539  
VdrvConfigType 1528  
VDrvControl 1540  
VdrvCtlOpCodeEnum 1530  
VDrvCustomControl 1542  
VDrvOpen 1543  
VDrvStatus 1545  
VDrvWrite 1545  
VerifyCallback SSL Attribute 2209  
vfprintf 913  
VFS Manager 1075  
VFSAnyMountParamType 103, 1076  
VFSCustomControl 1085  
VFSDirCreate 1087  
VFSDirEntryEnumerate 1088  
vfsErrBadData 1083  
vfsErrBadName 1083  
vfsErrBufferOverflow 1083  
vfsErrDirectoryNotFound 1083  
vfsErrDirNotEmpty 1083  
vfsErrFileAlreadyExists 1084  
vfsErrFileBadRef 1084  
vfsErrFileEOF 1084  
vfsErrFileGeneric 1084  
vfsErrFileNotFoundException 1084  
vfsErrFilePermissionDenied 1084  
vfsErrFileStillOpen 1084  
vfsErrIsADirectory 1084  
vfsErrNameShortened 1084  
vfsErrNoFileSystem 1084  
vfsErrNotADirectory 1084  
vfsErrVolumeBadRef 1084  
vfsErrVolumeFull 1085  
vfsErrVolumeStillMounted 1085  
VFSExportDatabaseToFile 1090  
VFSExportDatabaseToFileCustom 1091  
VFSExportProcPtr 1145  
vfsFileAttrArchive 1082  
vfsFileAttrDirectory 1082  
vfsFileAttrHidden 1082  
vfsFileAttrLink 1082  
vfsFileAttrReadOnly 1082  
vfsFileAttrSystem 1082  
vfsFileAttrVolumeLabel 1082  
VFSfileClose 1093  
VFSfileCreate 1094  
VFSfileDBGetRecord 1095  
VFSfileDBGetResource 1097  
VFSfileDBInfo 1099  
VFSfileDelete 1102  
VFSfileEOF 1103  
VFSfileGetAttributes 1104  
VFSfileGetDate 1105  
VFSfileOpen 1106  
VFSfileRead 1108  
VFSfileReadData 1109  
VFSfileRename 1111  
VFSfileResize 1113  
VFSfileSeek 1114  
VFSfileSetAttributes 1115  
VFSfileSetDate 1117  
VFSfileSize 1118

---

vfsFilesystemType\_AFS 1080  
vfsFilesystemType\_EXT2 1080  
vfsFilesystemType\_FAT 1080  
vfsFilesystemType\_FFS 1080  
vfsFilesystemType\_HFS 1080  
vfsFilesystemType\_HFSPlus 1080  
vfsFilesystemType\_HPFS 1080  
vfsFilesystemType\_MFS 1080  
vfsFilesystemType\_NFS 1080  
vfsFilesystemType\_Novell 1080  
vfsFilesystemType\_NTFS 1080  
vfsFilesystemType\_VFAT 1080  
VFSFileTell 1119  
VFSFileWrite 1120  
VFSGetDefaultDirectory 1121  
vfsHandledStartPrc 102  
vfsHandledUIAppSwitch 102  
VFSImportDatabaseFromFile 1123  
VFSImportDatabaseFromFileCustom 1124  
VFSImportProcPtr 1146  
VFSInstallFSLib 1127  
vfsModeCreate 1081  
vfsModeExclusive 1081  
vfsModeLeaveOpen 1081  
vfsModeRead 1081  
vfsModeReadWrite 1081  
vfsModeTruncate 1081  
vfsModeVFLayerOnly 1081  
vfsModeWrite 1081  
vfsMountClass\_POSE 1083  
vfsMountClass\_Simulator 1083  
vfsMountClass\_SlotDriver 1083  
VFSPOSEMountParamType 1078  
VFSRegisterDefaultDirectory 1128  
VFSRemoveFSLib 1130  
VFSSlotMountParamType 1077  
VFSUnregisterDefaultDirectory 1131  
vfsVolumeAttrHidden 1082  
vfsVolumeAttrReadOnly 1082  
vfsVolumeAttrSlotBased 1082  
VFSVolumeEnumerate 1132  
VFSVolumeFormat 1134  
VFSVolumeGetLabel 1137

VFSVolumeInfo 1138  
VFSVolumeMount 1139  
VFSVolumeSetLabel 1142  
VFSVolumeSize 1143  
VFSVolumeUnmount 1144  
Viewer application 2318  
virtual character 1913  
virtual driver 2321  
virtual driver functions 1538  
virtual driver queue functions 1546  
Virtual File System Manager 1075  
voltage warning threshold 963, 965  
VolumeInfoType 1078  
vsprintf 914  
vsprintf (StrVPrintF) 936

## W

WakeUpHandlerProc 1590  
WakeUpHandlerProcPtr 1589  
WbufSize SSL Attribute 2210  
Web Clipping Application Viewer 2318  
wiCmd... constants 1889  
WiCmdEnum 1888  
WinClipRectangle 1164  
WinCopyRectangle 1165  
WinCreateBitmapWindow 1166  
WinCreateOffscreenWindow 1168  
WinCreateWindow 1170  
WinDeleteWindow 1171  
WinDirectionType 1229  
WinDisplayToWindowPt 1172  
window list 287  
Window.h 1147  
WindowFlagsType 1155  
WindowFormatType 1157, 1168  
windows 1147–1245  
    active window 68  
    structure 1157  
WindowType structure 1157  
WinDrawBitmap 1172  
WinDrawChar 1173  
WinDrawChars 1174  
WinDrawGrayLine 1175

---

WinDrawGrayRectangleFrame 1176  
WinDrawInvertedChars 1176  
WinDrawLine 1177  
WinDrawOperation 1160  
WinDrawPixel 1178  
WinDrawRectangle 1178  
WinDrawRectangleFrame 1179  
WinDrawTruncChars 1180  
winEnterEvent 68, 69, 313  
WinEraseChars 1181  
WinEraseLine 1182  
WinErasePixel 1182  
WinEraseRectangle 1183  
WinEraseRectangleFrame 1184  
WinEraseWindow 1184  
winExitEvent 69  
WinFillLine 1185  
WinFillRectangle 1185  
WinGetActiveWindow 1186  
WinGetBitmap 1186  
WinGetClip 1188  
WinGetCoordinateSystem 1188  
WinGetDisplayExtent 1189  
WinGetDisplayWindow 1189  
WinGetDrawWindow 1190  
WinGetFirstWindow 1191  
WinGetFramesRectangle 1191  
WinGetPattern 1192  
WinGetPatternType 1193  
WinGetPixel 1193  
WinGetPixelRGB 1194  
WinGetSupportedDensity 1195  
WinGetWindowBounds 1187  
WinGetWindowExtent 1196  
WinGetWindowFrameRect 1197  
WinGlueDrawChar 1174, 1894  
WinGlueDrawTruncChars 1181, 1894  
WinGlueGetFrameType 1923  
WinGlueSetFrameType 1923  
WinHandle 1162  
WinIndexToRGB 1197  
WinInvertChars 1198  
WinInvertLine 1199  
WinInvertPixel 1199  
WinInvertRectangle 1200  
WinInvertRectangleFrame 1201  
WinLineType 1162  
WinLockInitType 1223  
WinModal 1201  
WinPaintBitmap 1202  
WinPaintChar 1203  
WinPaintChars 1204  
WinPaintLine 1205  
WinPaintLines 1205  
WinPaintPixel 1206  
WinPaintPixels 1207  
WinPaintRectangle 1207  
WinPaintRectangleFrame 1208  
WinPaintRoundedRectangleFrame 1209  
WinPaintTiledBitmap 1210  
WinPalette 89, 1211  
WinPopDrawState 1213  
WinPtr 1163  
WinPushDrawState 1214  
WinResetClip 1214  
WinRestoreBits 1215  
WinRGBToIndex 1215  
WinSaveBits 1217  
WinScaleCoord 1218  
WinScalePoint 1219  
WinScaleRectangle 1220  
WinScreenGetAttribute 1221  
WinScreenLock 1223  
WinScreenMode 89, 1224  
WinScreenModeOperation 1224  
WinScreenUnlock 1229  
WinScrollRectangle 1229  
WinSetActiveWindow 68, 1230  
WinSetBackColor 1231  
WinSetBackColorRGB 1232  
WinSetBounds 1233  
WinSetClip 1233  
WinSetCoordinateSystem 1234  
WinSetDrawMode 1234  
WinSetDrawWindow 1235  
WinSetForeColor 1236

---

WinSetForeColorRGB 1237  
WinSetPattern 1238  
WinSetPatternType 1238  
WinSetTextColor 1239  
WinSetTextColorRGB 1240  
WinSetUnderlineMode 1241  
WinUnscaleCoord 1241  
WinUnscalePoint 1242  
WinUnscaleRectangle 1243  
WinUseTableIndexes 1212

WinValidateHandle 1244  
WinWindowToDisplayPt 1245  
wireless internet feature set 2318  
WirelessIndicator.h 1888  
word wrap 734  
write callback function 1354  
WriteBlock 1548  
WriteBufPending SSL Attribute 2210  
WriteByte 1548  
WriteProc 1318, 1354

